2016

An Architecture for Configuring an Efficient Scan Path for a Subset of Elements

Arash Ashrafi
Louisiana State University and Agricultural and Mechanical College, arash.ashrafi90@gmail.com

AN ARCHITECTURE FOR CONFIGURING AN EFFICIENT SCAN PATH FOR A SUBSET OF ELEMENTS

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering

in

The School of Electrical Engineering and Computer Science

by
Arash Ashrafi
B.Sc., University of Tehran, 2013
May 2016

Acknowledgments

I would like to express my deepest and most sincere gratitude to my advisor Dr. Ramachandran Vaidyanathan for his patience, support and excellent guidance. Without his insightful and innovative ideas, this thesis could not have been accomplished.

I would also like to acknowledge my committee members Dr. Jerry Trahan and Dr. David Koppelman for their valuable remarks on my thesis and also the tremendous knowledge and insight I gained in their classes: *Structure of Computers and Computations* and *Digital Design Using HDLs*. My gratitude is extended to all my teachers in my graduate studies for their devoted instruction.

A special gratitude and love goes to my family. I thank my parents and my brother for their endless support and encouragement without which I may never have gotten to where I am today. Finally, I want to express my deepest love and gratitude to the music of my life, Leila, for her endless support and for always being there for me during the most difficult times of writing this thesis.

Table of Contents

Acknowledgments ii

List of Tables v

List of Figures vi

Abstract ix

1 Introduction 1

2 Problem and Approach 9
2.1 Scanning in Bits 9
2.2 FPGA Partial Reconfiguration (PR) 9
2.3 Traditional Method 10
2.4 Proposed Approach 11

3 The Basic Network 16
3.1 Basic Network Data Plane 16
3.2 Basic Network Control Plane 20

4 The Shortcuts Network 26
4.1 Shortcuts Network Data Plane 27
4.2 Selecting the Right Path 28
4.3 Shortcuts Network Control Plane 31

5 Clock Recommendation 37
5.1 Random Distribution 37
5.2 Contiguous Distribution 39
5.3 Clock Recommendation 41

5.4 Clock Generation with Shortcuts 45

6 Header Network 47
6.1 Determining the Header Leaf 47
6.2 Constructing the Header Path 52

7 Synthesis and Modeling 55
7.1 Synthesis Architecture and Design Flow 56
7.2 Synthesis Results 61
7.3 Modeling Stage 63
7.4 Comparison of Network Costs 71

8 Concluding Remarks 74

References 76

Vita 80

List of Tables

1.1 A summary of the results in this thesis for constructing a scan path of k elements out of n elements. 8

3.1 Control plane operation; "$*$" indicates a don't care. 21

4.1 Configurations of a leaf switch 29

7.1 Example report generated by the RTL Compiler Tcl script for the Basic Network with $n = 16$. 58

7.2 A part of the final report generated at the end of the synthesis stage. 70

List of Figures

1.1 An illustration showing the effect of frame granularity on partial reconfiguration. 3

2.1 A 6-bit scan of width (a) $w = 1$ and (b) $w = 2$ pins. 10

2.2 A schematic illustrating the primary and secondary scan paths 13

2.3 A schematic of the scan path 14

2.4 Structure of the proposed architecture 14

3.1 An example path in the Basic network 17

3.2 The different configurations of the internal nodes 18

3.3 Internal structure of data plane of node x. 19

3.4 An example of the two proposed methods for implementing frames with more than one configurable bits 20

3.5 Internal structure of the control node x. 22

3.6 Block diagram of the Basic Network of size $n = 4$ 24

4.1 An example path in the Shortcuts Network. 26

4.2 Internal structure of the data plane of a leaf switch x. 28

4.3 The procedure of selecting the right path between two adjacent elements of $\mathbb{C}$ 30

4.4 The FSM to select between the tree edges and shortcut edges. 32

4.5 The internal structure of node x in the control plane of the Shortcuts Network. 35

4.6 The internal structure of leaf switch x in the control plane of the Shortcuts Network. 36

5.1 An schematic illustrating the possible positions of k contiguous elements. 40

5.2 The internal structure of node x in the Clock Recommendation Network. 43

5.3 Block diagram of the Clock Recommendation Network of size $n = 8$. 44

6.1 An example illustrating prefix OR of n bits by cascading $n - 1$ OR gates. 48

6.2 An illustration of EX-OR of n bits. 48

6.3 An example illustrating prefix OR of n bits. 49

6.4 An example illustrating the non-recursive implementation of a prefix OR of n bits. 51

6.5 Internal structure of node x and leaf i in the prefix OR tree. 52

6.6 Block diagram of the Header Network with $n = 4$. 53

6.7 An example of establishing the header path π_0 using tristate buffers. 54

7.1 An illustration of the design flow. 57

7.2 Flow chart of the algorithm used in the RTL Compiler Tcl script to optimize the networks with respect to the clock period (i.e. "time-optimized"). 59

7.3 Example schematic for the Basic Network with $n = 16$. 60

7.4 Example layout for the Shortcuts Network with $n = 16$. 62

7.5 Clock period (T) results for the Basic Network as a function of size n. 64

7.6 Clock period (T) results for the Basic Network as a function of size $\log n$ (logarithmic x-axis). 64

7.7 Area (A) results for the Basic Network as a function of size n. 65

7.8 Power (P) results for the Basic Network as a function of size n. 65

7.9 Clock period (T) results for the Basic Network with clock recommendation as a function of size n. 66

7.10 Clock period (T) results for the Basic Network with clock recommendation as a function of $\log n$ (logarithmic x-axis). 66

7.11 Area (A) results for the Basic Network with clock recommendation as a function of size n. 67

7.12 Power (P) results for the Basic Network with clock recommendation as a function of size n. 67

7.13 Clock period (T) results for the Shortcuts Network as a function of size n. 68

7.14 Clock period (T) results for the Shortcuts Network as a function of $\log n$ (logarithmic x-axis). 68

7.15 Area (A) results for the Shortcuts Network as a function of size n. 69

7.16 Power (P) results for the Shortcuts Network as a function of size n. 69

7.17 Clock period T as a function of n for the synthesized networks. 72

7.18 Area A as a function of n for the synthesized networks. 73

7.19 Power P as a function of n for the synthesized networks. 73

Abstract

Field Programmable Gate Arrays (FPGAs) have many modern applications. A feature of FPGAs is that they can be reconfigured to suit the computation. One such form of reconfiguration, called partial reconfiguration (PR), allows part of the chip to be altered. The smallest part that can be reconfigured is called a frame. To reconfigure a frame, a fixed number of configuration bits are input (typically from outside) to the frame.

Thus PR involves (a) selecting a subset $\mathbb{C} \subseteq \mathbb{S}$ of k out of n frames to configure and (b) inputting the configuration bits for these k frames. The, recently proposed, MU-Decoder has made it possible to select the subset $\mathbb{C}$ quickly. This thesis involves mechanisms to input the configuration bits to the selected frames.

Specifically, we propose a class of architectures that, for any subset $\mathbb{C} \subseteq \mathbb{S}$ (set of frames), constructs a path connecting only the k frames of $\mathbb{C}$ through which the configuration bits can be scanned in. We introduce a Basic Network that runs in $\Theta(k \log n)$ time, where k is the number of frames selected out of the total number n of available frames; we assume the number of configuration bits per frame is constant. The Basic Network does not exploit any locality or other structure in the subset of frames selected. We show that for certain structures (such as frames that are relatively close to each other) the speed of reconfiguration can be improved. We introduce an addition to the Basic Network that suggests the fastest clock speed that can be employed for a given set of frames. This enhancement decreases configuration time to $O(k \log k)$ for certain cases. We then introduce a second enhancement, called shortcuts, that for certain cases reduces the time to an optimal $O(k)$. All the proposed architectures require an optimal $\Theta(n)$ number of gates.

We implement our networks on the CAD tools and show that the theoretical predictions are a good reflection of the network's performance.

Our work, although directed to FPGAs, may also apply to other applications; for example hardware testing and novel memory accesses.

Chapter 1
Introduction

Reconfigurable devices, notably Field-Programmable Gate Arrays (FPGAs) have gravitated to a mainstream role in the modern computing landscape, for example, in image and video processing [1, 2, 3, 4], networking and network security [5], embedded systems [6, 7, 8], high-speed trading [9, 10], and as accelerators in high-performance computing [11, 12, 13, 14]. One important feature of some FPGAs is (dynamic) partial reconfiguration, where a portion of the FPGA can be reconfigured during run-time [15, 16]. A key determinant of the usefulness of dynamic partial reconfiguration is its speed [17]. In this work we propose an approach and a set of methods that could significantly speed up partial reconfiguration.

For the purpose of partial reconfiguration, the configurable fabric of an FPGA is divided into *frames* [18]; in this thesis we will use the term "frame" to mean the smallest unit that can be partially reconfigured; that is, even if a portion of a frame is to be reconfigured, the entire frame must be reconfigured. In an FPGA each frame is divided into configuration units and the number of these units determines the size of that frame. Frame size can (theoretically) be from one single configuration element to several thousand units per frame [19]. Configuration elements can be thought as ON/OFF switches that are reconfigured to change the function and/or connectivity of the chip. A frame is reconfigured by applying a configuration bit to each of its configurable elements. The configuration bits originate outside the frame (often outside the chip). Since the number of configuration bits needed (even for a single frame) is much larger than the number of wires/pins available to convey them, a large number of these bits are serially shifted in through a scan path traversing the configurable elements of a frame.

Typically a software interface maps each user-defined partially reconfigurable (PR) module to a set of frames. Usually, at most one PR module may be associated with

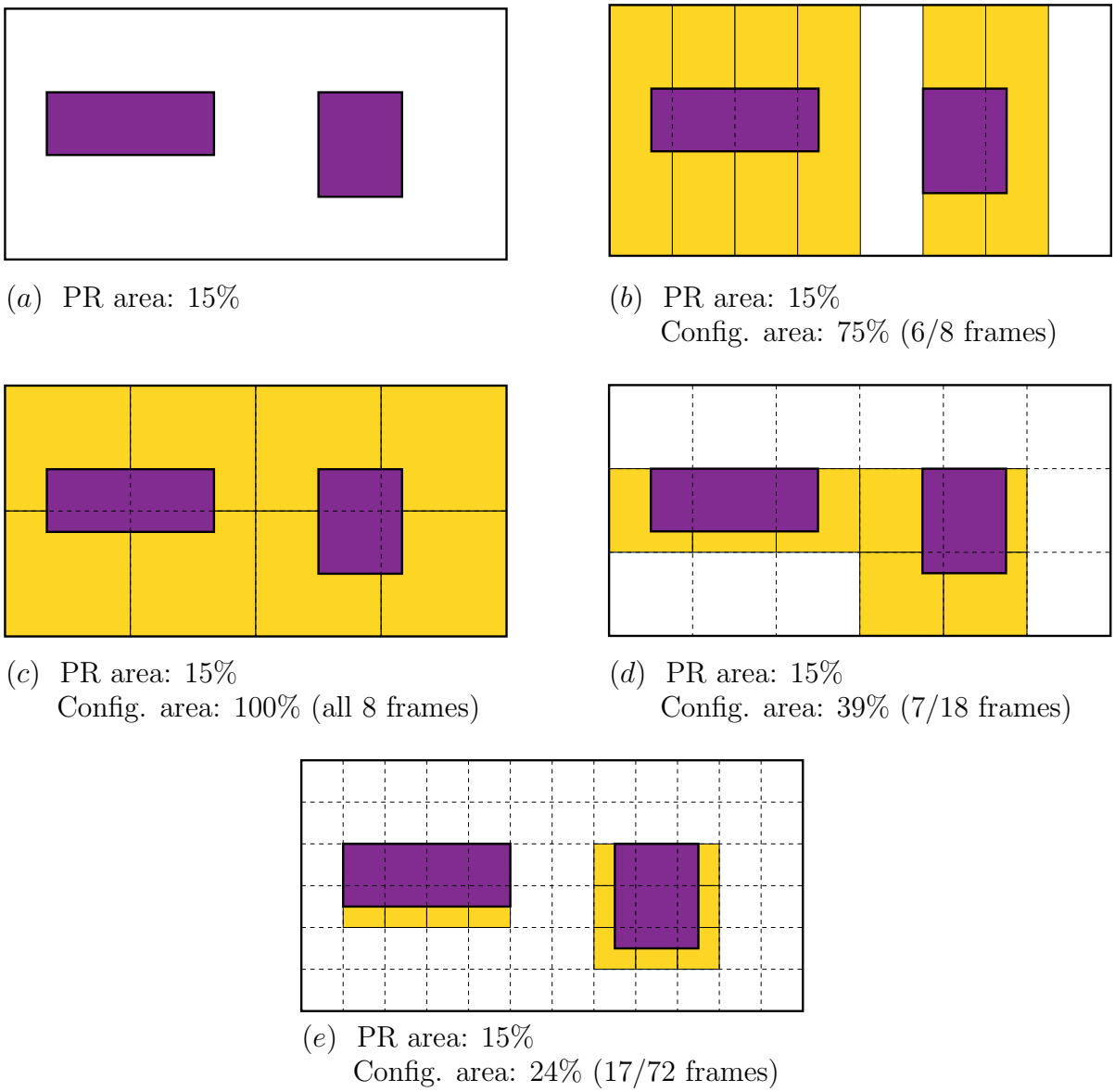
a frame; however, a single PR module may span across multiple frames. This, coupled with the fact that frame sizes and shapes many not be well-matched to the PR module's ideal proportions, often causes a somewhat loose fit between the PR modules and frames encompassing them. Consequently, many more elements may need to be reconfigured than are directly associated with the PR module.

In general, the smaller the frame (lower granularity/higher resolution), the better the fit between frames and the PR region. Figure 1.1 illustrates the effect of frame's shape and size on the efficiency of the reconfiguration. The PR region colored as blue, is constant (15%) for all the cases. It is important to note that although cases (b) and (c) have the same number of total frames (8 frames), the total configuration area is different in each case; solely due to difference in the shape of the frames. Clearly as the size of a frame decreases, the overall configuration area decreases while keeping the PR area constant (compare cases (c), (d), and (e).)

Thus, using smaller frames could alleviate the problem of such an "unfocused" reconfiguration. However, this would result in a larger number of frames being selected. For all practical purposes, a typical FPGA selects and reconfigures one frame at a time; in this approach, using a large number of small frames would not be economical.

The MU-Decoder [21, 22], proposed recently, allows for multiple frames to be selected simultaneously. It uses knowledge of the computation to speedily (in $O(\log n)$ time) select a subset of frames for reconfiguration. In particular, if a subset $\mathbb{C}$ with k elements is to be selected out of n frames of $\mathbb{S}$ for reconfiguration, then the MU-Decoder could select all k frames in a handful of iterations; in contrast, the standard method (using a one-hot-decoder) uses k iterations to select all frames. We now illustrate the benefit of simultaneously configuring small frames.

Suppose an FPGA has f configurable bits divided into n_1 frames (with $d_1 = \frac{f}{n_1}$ bits per frame). Suppose an instance of partial reconfiguration changes b configurable elements and on an average a fraction α_1 of a frame is used. Then the b configurable elements occupy



(*a*) PR area: 15%

(*b*) PR area: 15%
Config. area: 75% (6/8 frames)

(*c*) PR area: 15%
Config. area: 100% (all 8 frames)

(*d*) PR area: 15%
Config. area: 39% (7/18 frames)

(*e*) PR area: 15%
Config. area: 24% (17/72 frames)

Figure 1.1: An illustration showing the effect of frame granularity on partial reconfiguration. The two colored regions show the area that needs partial reconfiguration which remains the sames for all cases. The shape and size of the frames is altered for each case and the reconfigured area and the number of reconfigured frames are reported. Figure is taken with permission from [20].

$k_1 = \dfrac{b}{\alpha_1 d_1}$ frames. Let c_1 be the overhead for configuring each frame (frame address, checksum bits etc.). Then in a regimen that accesses one frame at a time the bitstream size is roughly

$$B_1 = (d_1 + c_1)\, k_1 = d_1 \left(\frac{b}{\alpha_1 d_1} \right) + c_1 k_1 = \frac{b}{\alpha_1} + c_1 k_1.$$

On the other hand if the f bits are divided into a much larger number n_2 of frames each with $d_2 = \dfrac{f}{n_2}$ bits, the the average utilization of the frame in partial reconfiguration will be a much higher fraction α_2. Moreover, if all $k_2 = \dfrac{b}{\alpha_2 d_2}$ frames are configured simultaneously with one-time overhead c_2, then the bitstream size for partial reconfiguration here is roughly

$$B_2 = d_2 k_2 + c_2 = d_2 \left(\frac{b}{\alpha_2 d_2} \right) + c_2 = \frac{b}{\alpha_2} + c_2.$$

To compare the two cases we first observe that c_2 and $c_1 k_1$ are comparable. This is because the overhead has an approximately linear relationship with the size of the bitstream. Now, B_1 can be significantly larger than B_2 based on how much the number of frames has increased and their size decreased. That is,

$$B_1 - B_2 \cong b \left(\frac{1}{\alpha_1} - \frac{1}{\alpha_2} \right), \quad \alpha_1 \ll \alpha_2$$

The bitstream size is a good measure of the time needed for partial reconfiguration [17]. That is $B_1 - B_2$ is a measure of how much faster the system with smaller frame size can be. The difference in the two approaches could be further amplified by the fact that the MU-Decoder [21, 22], used to simultaneously access all frames requiring reconfiguration has small time overheads (compared to multiple single-frame accesses in the conventional approach).

To utilize this capability of the MU-Decoder to quickly select multiple frames for reconfiguration, however, the configuration bitstream should be directed to only those frames that need reconfiguration. Specifically if a subset of k frames (from a total of n frames)

need to be reconfigured, then the problem is to generate a scan path that weaves through just the k elements requiring reconfiguration. In addition, this path must be generated quickly, and it must allow fast clocking of the configuration bitstream. Finally, the scan path must be generated automatically in hardware, given just the subset of frames to be reconfigured.

Contribution of this work: In this thesis, we propose several networks that generate such scan paths. For a system with n frames, all of these networks have $O(n)$ size[1] (asymptotically the best possible). In broad terms the networks we propose work as follows. Given a subset of frames to reconfigure, the network autonomously constructs a path through the selected frames, in some cases also determines the maximum clocking speed for this path. This phase of the networks' operation is called "preprocessing." In the next phase (configuration phase) the configuration bits of the k selected frames are input. The simplest network (the Basic Network) scans in the bitstream in $O(k \log n)$ time, where k is the number of frames to be configured. More complex networks (those with Clock Recommendation and Shortcuts) run in $O(\log n \log\log\log n + kT_0)$ time, where T_0 can be quite small, depending on the distribution of the k elements over the n elements; here $\log n \log\log\log n$ is the preprocessing time and kT_0 is the time for inputting bits. These networks also produce a clock speed recommendation (consistent with T_0) that the bitstream source can use. Table 1.1 at the end of this chapter summarizes the results in this thesis. It should be noted that typically $\log n \ll k \ll n$. Therefore, the preprocessing times of Table 1.1 or the $O(\log n)$ delay of the MU-Decoder are not very significant overheads when compared to the $\Omega(k)$ reconfiguration time. In particular, with $\log n \ll k \ll n$ the PR time converges to $O(k)$ for the networks with Clock Recommendation and Shortcuts which is asymptotically optimal. Moreover, if the selected frames for reconfiguration are close to each other (as is usually the case), the clock period T_0 becomes constant. We also present results from an

[1]In this thesis we assume a gate with constant fan-in and fan-out has constant delay and size. Likewise a flip-flop or latch is also assumed to have constant delay and size. The delay of a network is the delay along its longest path. The size of the network is the number of gates and flip-flops in it.

implementation on the the Cadence suite of CAD tools to back our theoretically derived performance estimates (reconfiguartion time and hardware cost).

Related work: There is not much related work on selecting subset of frames, other than the one-hot-decoder [23] (which is a folklore logic component) and the MU-Decoder [21]. On scan paths, there is a lot of work in the context of hardware testing [24, 25, 26, 27]; however, generate scan paths that are relatively static and are not suitable for the $\binom{n}{k}$ possibilities associated with the problem we address, where n is in millions and k potentially in hundreds. More broadly, reconfigurable computing itself is discussed in detail [28]. It has recognized the benefit of fast reconfiguration from the power of theoretical models to self reconfiguration [29, 30, 31]. Other ideas such as configuration compression [32, 33, 34] also reduce the time to input bits, but spend additional time to expand the bits out before actual reconfiguration is performed. We are not aware of any work on a dynamically configurable scan path, as proposed in this work.

Organization of thesis In the next chapter we define the problem and describe the overall structure of our solution. In Chapter 3 we describe the Basic Network constructed on an underlying binary tree. In Chapter 4 we augment the tree in the Basic Network with shortcut connections across leaves. This admits faster clock speeds for the configuration bitstream, without significant network cost. Chapter 5 deals with the clock recommendation network. We first analyze two extreme situations for the distribution of the k elements to be configured among the set of n elements. For the first of these, a random distribution, there is not much improvement in the clock speed. The second, contiguous distribution, admits (on an average) a clock speed that is independent on n. Chapter 5 introduces a hardware method to output a clock speed recommendation that is customized to the particular set of k elements to be configured. Chapter 6 details the Header Network that is essential to exploiting faster clock speeds; Chapter 2 explains the need for the Header Network. Chapter 7 details the implementation of the networks, presents simulation results,

and derives model equations for network performance. In Chapter 8 we summarize our results and identify directions for future work.

Table 1.1: A summary of the results in this thesis for constructing a scan path of k elements out of n elements.

Network	Distribution	Network cost	Preprocessing time	Scan time per bit	Partial reconfiguration time (total)
Basic Network (Chapter 3)		$O(n)$	$O(\log n)$	$O(\log n)$	$O(k \log n)$
Clock Recommendation Network (Chapter 5)	Random (average time)	$O(n)$	$O(\log n \log\log\log n)$	$O(\log n)$	$O(\log n \log\log\log n + k \log n)$
	Contiguous (avg. time)	$O(n)$	$O(\log n \log\log\log n)$	$O(\log k)$	$O(\log n \log\log\log n + k \log k)$
Shortcuts Network (Chapter 4)	Random (avg. time)	$O(n)$	$O(\log n \log\log\log n)$	$O(\log n)$	$O(\log n \log\log\log n + k \log n)$
	Contiguous	$O(n)$	$O(\log n \log\log\log n)$	$O(1)$	$O(\log n \log\log\log n + k)$

Chapter 2
Problem and Approach

In this chapter we describe the problem addressed in this thesis and outline the traditional approach and the proposed solution.

Consider an FPGA with n frames with d bits per frame (each of which requiring one configuration bit). Let $\mathbb{S} = \{0, 1, 2, \ldots, n-1\}$ be the set of frames. Given a subset $\mathbb{C} \subseteq \mathbb{S}$ of k frames, the goal is to configure the frames of $\mathbb{C}$ by sending in kd configuration bits from outside the chip through a small number of input lines (bandwidth limited interface). This "partial reconfiguration" must be done as quickly as possible.

2.1 Scanning in Bits

A frame may be viewed as a collection of d flip-flops each holding one configuration bit; we will call them configuration flip-flops. One way to input these d configuration bits through a single input pin is to connect the configuration flip-flops as a d-bit shift register and then clock in the d configuration bits through the input. We will call this a *d-bit scan of width 1*. This can be generalized to use $w \geq 1$ pin as follows. Let $m = d/w$ be an integer. Simply send the bits in through $w = d/m$ pins each scanning an m-bit scan of width 1. This, more general approach is called a *d-bit scan of width w*. Figure 2.1 shows 6-bit scans of width 1 and 2. The key feature of a scan is that there must be a logical path that connects the appropriate configuration flip-flops as a shift register for a d-bit scan of width w; we will call this set of shift registers as the d-bit *scan-path* of with w.

2.2 FPGA Partial Reconfiguration (PR)

Coming back to an FPGA with n frames and d bits per frame, partial reconfiguration of an FPGA amounts to reconfiguring the configuration flip-flops of a selected subset of

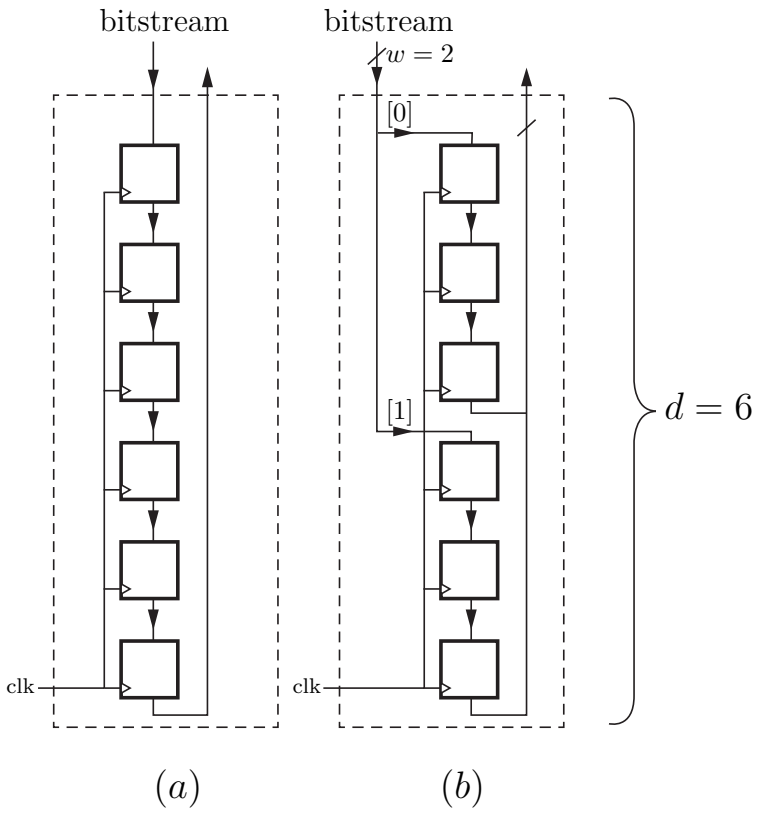


Figure 2.1: A 6-bit scan of width (a) $w = 1$ and (b) $w = 2$ pins. Dashed line shows the boundary of a frame.

k frames (where $1 \leq k \leq n$). We seek to perform this quickly and cost-effectively. This involves the following two steps:

(a) Selecting a subset $\mathbb{C}$ of k frames for reconfiguration;

(b) Inputting kd configuration bits for the selected frames.

We now describe the traditional approach to partial reconfiguration and contrast it with the proposed approach.

2.3 Traditional Method

The traditional method for partial reconfiguration uses a one-hot decoder to select one frame at a time. Assuming that the FPGA can use only one pin for partial reconfiguration (i.e. $w = 1$), a d-bit bitstream is then shifted in in d clock cycles through a scan path inside the selected frame. When all bits are in their intended location (flip-flops), a signal

instructs the frame to read the bits for reconfiguration. The data path from the input pin to the selected frame is similar to those used in memory to access an addressed location [35]. If the one-hot decoder requires t_{sel} for selecting a frame, then the total time needed for reconfiguring k frames with this method is

$$k\left(t_{sel}+d\right). \tag{2.1}$$

With w pins, PR can be done in $\frac{d}{w}$ clock cycles as explained earlier. Here, the total time needed to perform PR on k frames is

$$k\left(t_{sel}+\frac{d}{w}\right). \tag{2.2}$$

In an FPGA, the total number of frames, n, can be in order of millions and the number of reconfigured frames k in order of thousands. The number of pins of an FPGA, on the other hand, is very limited ($w \ll k$) due to packaging limitations. Therefore, it is not possible to perform PR on a substantial number of frames in parallel (all at the same time). On the other hand, we have seen that allowing w pins does not fundamentally change the standard process of PR and we can reconfigure the internal flip flops in parallel. As a result, without loss of generality, we will assume that the FPGA has one pin allocated ($w = 1$) for partial reconfiguration.

2.4 Proposed Approach

The proposed approach uses a MU-Decoder instead of a one-hot-decoder to simultaneously select a subset of k frames to reconfigure. The operation of the MU-Decoder is described in earlier work [22]. In this thesis, we assume the availability of the MU-Decoder and describe the problem of inputting configuration bits.

This capability of the MU-Decoder to quickly select a subset of frames to reconfigure can be utilized to speed up PR by eliminating the overheads of selecting each frame, one

by one, using a one-hot decoder. This, however, creates the problem of delivering the reconfiguration bitstream to only those selected frames. A static method to send bits to the selected frames amounts to total reconfiguration as the unselected frames would not be distinguished from the selected.

The proposed method involves an architecture constructs a dynamic physical scan path that "weaves" through only those frames that have been selected. Now we describe the proposed approach in detail.

Let $\mathbb{S} = \{0, 1, \cdots, n-1\}$ be a set of configurable elements (frames) and let $\mathbb{C} \subseteq \mathbb{S}$ be a k-frame subset requiring partial reconfiguration. The strategy is to construct a "primary" k-bit scan path that traverses only the selected frames (see Figure 2.2); we will detail this later. For simplicity and without loss of generality, let us assume that each frame requires w configuration bits (w is the width of the scan path). To accommodate $d > w$ configuration bits per frame of $\mathbb{C}$, we could use secondary scan d-bit scan path as shown in Figure 2.2. As noted earlier, the objective is to construct a circuit that connects (only) the k elements of $\mathbb{C}$ in a k-bit scan path in which elements of $\mathbb{C}$ represent flip-flops (sequential elements) and all hardware between flip-flops is combinational (see Figure 2.3). The circuit has k flip-flops (say $F_0, F_1, \cdots F_{k-1}$) connected serially with π_i as the combinational path from F_{i-1} (if it exists) to F_i; $0 \leq i < k$. Let t_i denote the delay of path π_i. Then ignoring flip-flop set-up/hold times and safety margins, the minimum time between two clock pulses for this circuit is

$$T_0 = \max\{t_0, t_1, \cdots, t_{k-1}\} \tag{2.3}$$

and the maximum clock frequency is $f = \frac{1}{T_0}$. On such a circuit, k bits can be scanned in in kT time. We will call T, the *delay* of the scan path. We will also call sets $\mathbb{S}$ and $\mathbb{C}$ as the *scan set* and the *configuration set*, respectively. We have assumed that $|\mathbb{S}| = n$ and $|\mathbb{C}| = k$.

We propose an architecture that, when given set $\mathbb{C} \subseteq \mathbb{S}$, configures itself to dynamically form a scan path with small delay that traverses the elements of $\mathbb{C}$. Figure 2.4 shows the

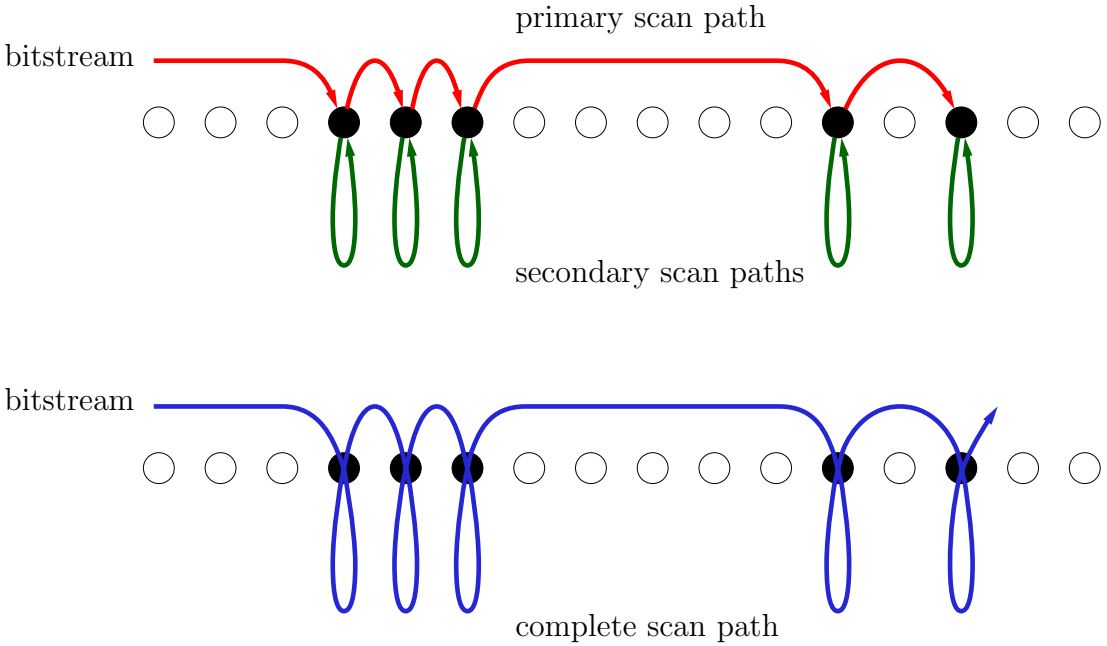


Figure 2.2: A schematic illustrating the primary and secondary scan paths. The paths marked as red and green in the upper schematic are the primary and secondary paths, respectively. The path marked as blue in the lower schematic is the complete scan path consisting of the primary and the secondary paths.

overall structure of the architecture. The *scan-path network* establishes the combinational paths $\pi_1, \pi_2, \cdots, \pi_{k-1}$ based on the scan set $\mathbb{C}$. In fact it can also establish π_0. However for the particular network proposed in Chapter 3, $T_0 = \Theta(\log n)$ and this causes the delay of the entire scan path to be $\Theta(\log n)$ because the length of all possible combinational paths in that network is $O(\log n)$. If this delay is acceptable, then the header path network is not necessary. Otherwise, the header path network establishes a fast path π_0 to the first element of $\mathbb{C}$. This, in turn, opens up the possibility of speeding up the configuration bitstream clock.

Each of the scan-path and header-path networks has two "planes," the data and control planes. The data plane is what is highlighted in Figure 2.4. This plane is entirely combinational (except for the configuration flip-flops) and has the path connecting the flip-flops corresponding to elements of $\mathbb{C}$, through which the configuration bitstream is to be scanned

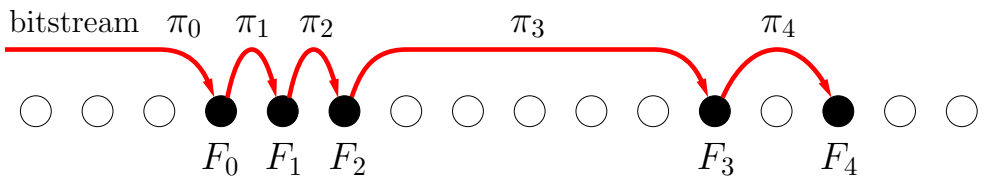


Figure 2.3: A schematic of the scan path for an example, in which $k = 5$ out of 16 elements have been selected; π_0–π_4 represent combinational paths.

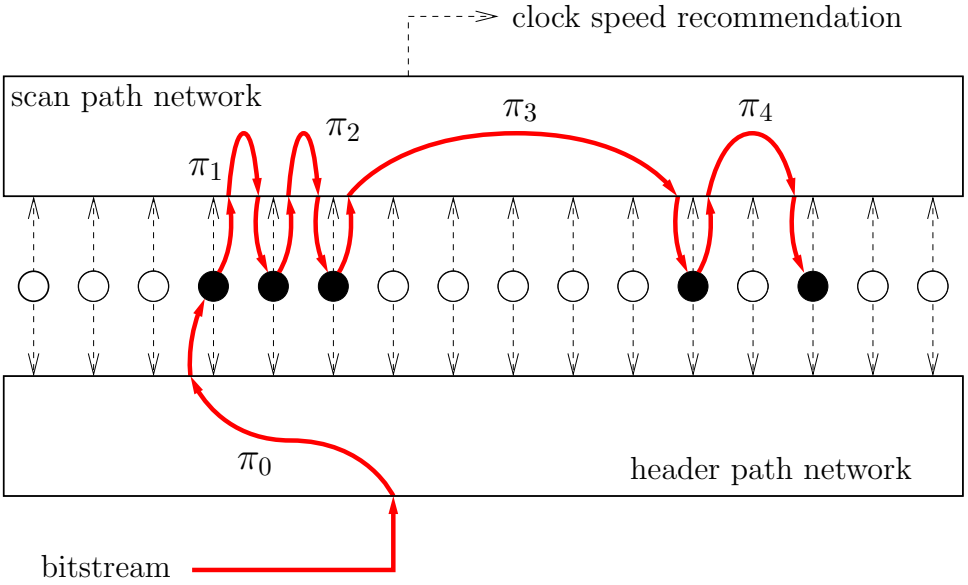


Figure 2.4: Structure of the proposed architecture; the example of Figure 2.3 has been used here. The data path is shown highlighted. Control connections are shown dashed.

in. The control plane primarily generates signals to set up the path(s) in the data plane. The control plane of the scan-path network may, optionally, also generate a clock speed recommendation based on the path-lengths in the data plane which will be covered later in Chapter 5. The control plane may contain sequential elements, but the configuration bitstream does not traverse this plane.

The main tasks performed by these networks are (a) configure header path π_0, (b) configure the scan path $\pi_1, \pi_2, \cdots, \pi_{k-1}$ (c) generate clock recommendation. The bitstream can be scanned in as soon as these actions are completed. While the header and scan paths can be established in parallel, the clock recommendation cannot be generated until the scan path has been established. This is because the clock rate depends on the delays of the

combinational segments of the scan path. If the three tasks above require t_h, t_s, t_c time, respectively (for header, scan and clock), then the minimum time before the bitstream can be scanned in is $\max\{t_h, t_s + t_c\}$. If the clock recommendation is to have clock cycle of duration T_0, then the total time needed for partial reconfiguration is

$$\max\{t_h, t_s + t_c\} + kT_0. \tag{2.4}$$

Here, $\max\{t_h, t_s + t_c\}$ is the preprocessing time and kT_0 is the time to scan in the k bits. Preprocessing time is the time to set up the scan path and generate the clock speed recommendation, but, in general, any time spent to initialize the network and prepare it for the data is considered as the preprocessing or initialization time. In this work, the total time to input kd configuration bits will be used as the performance measure for the proposed architectures. For certain cases, our method achieves the "optimal" $O(kd)$ time. The proposed architectures also have a gate cost of $\Theta(n)$ which is optimal considering that n configuration flip-flops are needed.

The various parts of the networks described in following chapters are coordinated by a "Master Controller" whose function includes the following:

(1) Start the preprocessing once a request to partially reconfigure is received after frames have been selected for reconfiguration; and

(2) Detect the end of preprocessing and send a signal to the configuration server to send in the configuration bits.

Chapter 3
The Basic Network

The Basic Network is the underlying architecture that makes the dynamic reconfiguration possible using our approach. Given a subset of frames to reconfigure, the Basic Network provides a physical path that "weaves" through only those frames that need reconfiguration. The configuration bitstream can then be shifted in through the path to reconfigure the selected frames. Later on, we will propose other modifications to the Basic Network (such as the networks with clock recommendation and shortcuts); therefore, the Basic Network is a crucial building block for our proposed designs. The Basic Network (as the other proposed networks) has two planes: the *data plane* and the *control plane*. As we discussed in the previous chapter, the data plane is responsible for delivering the configuration bits to the flip-flops corresponding to the selected frames and the control plane is responsible for providing the control signals to set up the data path in the data plane. We now describe the data and control planes.

3.1 Basic Network Data Plane

As described in Chapter 2, a key attribute of the network is to allow for paths connecting elements of all $2^n - 1$ possible non-empty subset of frames, while keeping the combinational delay of the path segment between frames low. To this end, the approach used is similar to that of Roy *et al.* [36]. Consider the binary tree shown in Figure 3.1. It consists of "internal nodes" and "leaf nodes" through the subset of which the final configuration bitstream is going to pass. In Figure 3.1, internal nodes are shown as triangles and leaf nodes as circles at the bottom. Leaves that have been selected for reconfiguration are highlighted. We will call them "active" leaf nodes.

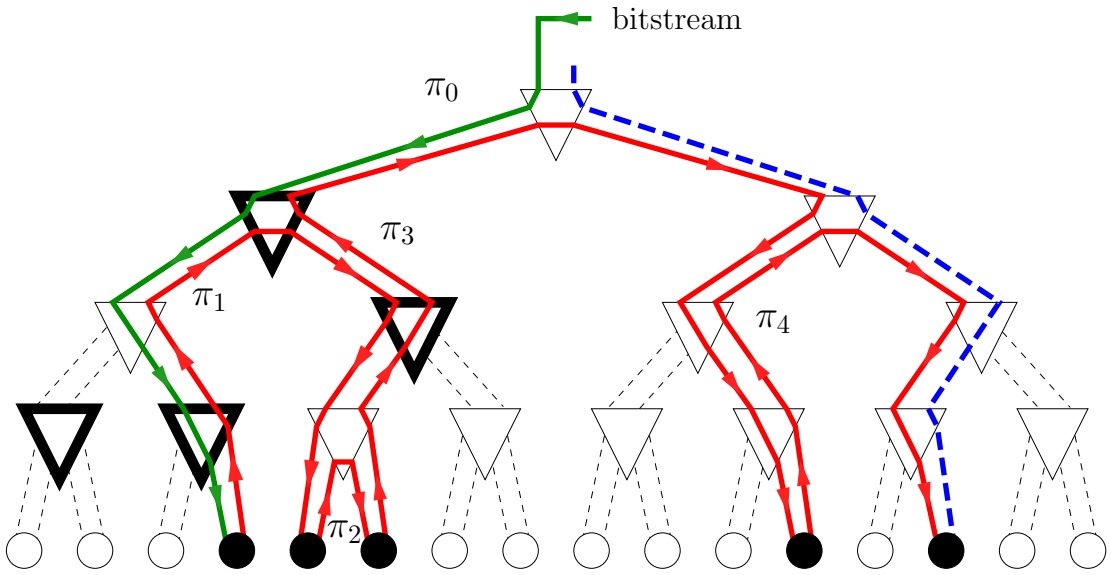


Figure 3.1: An example path in the Basic network. The underlying tree is shown dotted. The header path π_0 is shown separately (starting at the root). A "trailer" path terminating at the root is also shown dashed; this has no impact on the scan path delay. However, allowing this path restricts the number of internal configurations of a tree node to the ones shown in bold.

Each internal node x in the underlying tree has three pairs of ports, $p_i(x)$, $p_o(x)$, $\ell_i(x)$, $\ell_o(x)$, $r_i(x)$, $r_o(x)$ (see Figure 3.3); one pair connects to the parent of x and the remaining pairs connect to the left and right children of x. Let x be the left child of node y; that is, y is the parent of node x. Then port $p_i(x)$, that represents the input port of x from its parent y, connects from $\ell_o(y)$, the output port of y to its left child x. Similarly $p_o(x)$ connects to $\ell_i(y)$ and ports $\ell_i(x), \ell_o(x), r_i(x), r_o(x)$ to the left and right children of x. By internally connecting input ports to output ports within each node, a path can be established within the tree (much like buses are constructed on a Reconfigurable Mesh [28]). For example in Figure 3.1, the root internally connects p_i to ℓ_o for path π_0 to be constructed from the input bitstream down to root's left child.

The internal nodes of the Basic Network make internal connections to establish paths between leaves. Figure 3.2 shows the configurations of the internal nodes of the Basic Network needed for our purpose. Figure 3.2 (a) represents a node where descendants are not part of any path. Figure 3.2 (b) and (c) represent a situation where all selected frames

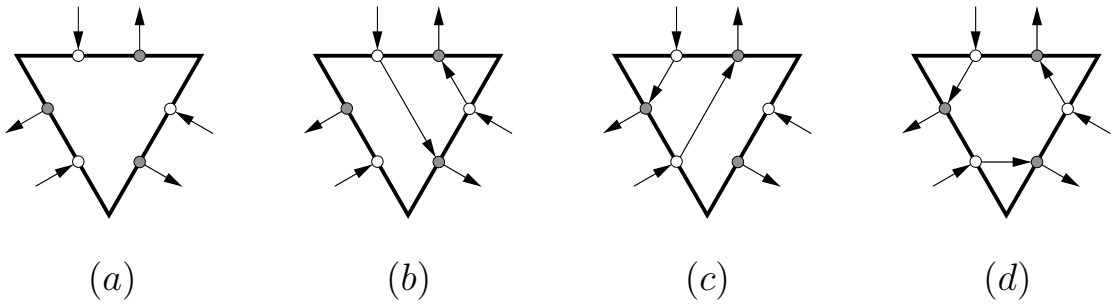


Figure 3.2: The different configurations of the internal nodes. These variations are shown in bold in Figure 3.1.

lie to one of the "sides" of the node. In Figure 3.2 (d) both sides of the node have selected frames, so the path needs to traverse the left side first and then cover the right before returning to the parent. It should be noted that other possible internal configurations (such as traversing from the parent to the right and then the left, or making a U-turn at an internal node) will not be needed for our approach (see Section 3.2).

Figure 3.3 shows a possible design for the internal structure of a tree node in the data plane that accommodates the four configurations of Figure 3.2. For an output port on the top (to parent) and right (to right child) "sides" of the node, there is a multiplexer that selects at most one of the two inputs from the other two sides of the node; this includes the possibility of selecting neither input port through a tri-state gate.

As shown in Table 3.1, by appropriately setting the control lines, $c_0(x)$, $c_1(x)$, $c_2(x)$, $c_3(x)$, and $c_4(x)$, the four configurations for the internal connections can be realized. It is clear that once the multiplexers and tri-state gates are properly configured, the scan path is ready (see Figure 3.1) to shift in the bitstream; we will show later in Section 3.2 that the path traverses only the selected leaves.

So far we have seen this tree structure constructs a datapath that is completely combinational. The leaf nodes at the bottom of the tree are the only sequential elements in this architecture. Each leaf node is implemented using a simple D flip-flop in order to be able to store the final values when the bitstream is completely shifted in; meaning the first configuration bit has reached the last active leaf of the network. Each stored bit will then

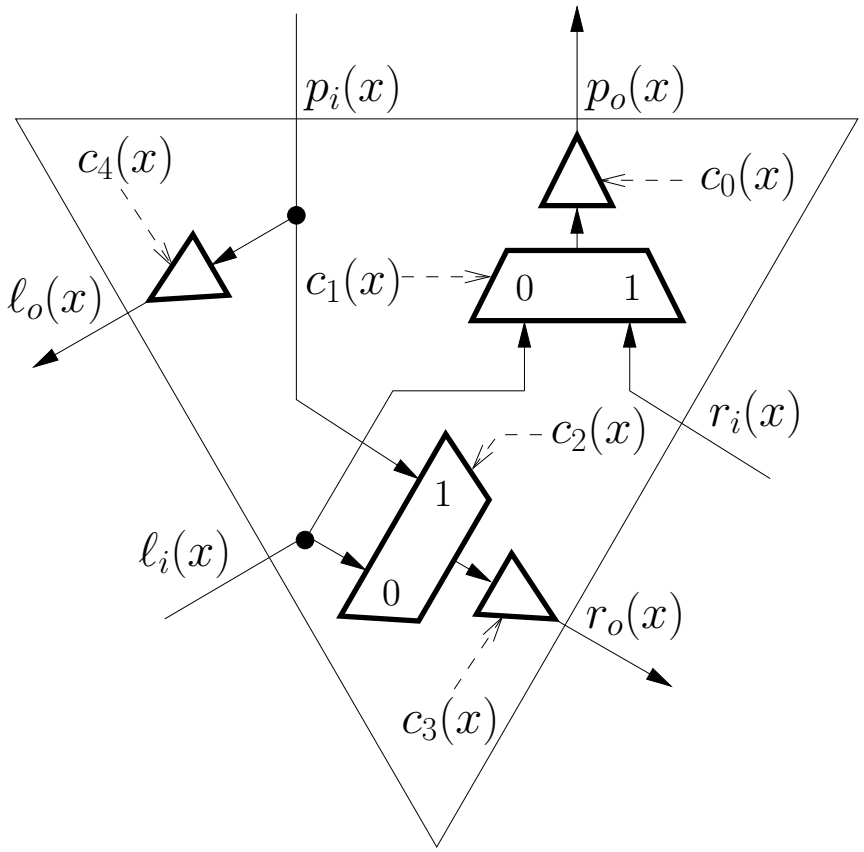


Figure 3.3: Internal structure of data plane of a tree node (data node x). The multiplexers and tri-state gates allow for the connections shown in Table 3.1.

reconfigure the corresponding frame. As a result, we might interchangeably use the term "frame" to refer to the leaf nodes and vice versa.

Recall that we assumed (without loss of generality) that each frame needs only one configuration bit. If a frame needs more than one configuration bit, we can still employ a similar idea as stated below. Let each leaf node require d configuration bits. Now, there are two options for sending in the configuration bitstream to a frame, while utilizing the same Basic Network architecture: the serial bits option and parallel bits option. Figure 3.4 shows these methods. In the first method, the configuration bits of each frame are placed one after another in a sequence and the bitsream entering each frame is serially shifted through in d clock cycles (see Figure 3.4 (a)). In order to accomplish reconfiguration, the length of the required bitstream in this case is $B_s = B \cdot d$, where B is the length of original bitstream. In the parallel method, the width of original bitstream has to increase to d bits

so they can be delivered in parallel, d at a time per clock cycle (Figure 3.4 (b)). Clearly, one could have a hybrid method where the d bits of a frame are sent b bits at a time where $1 \leq b \leq d$.

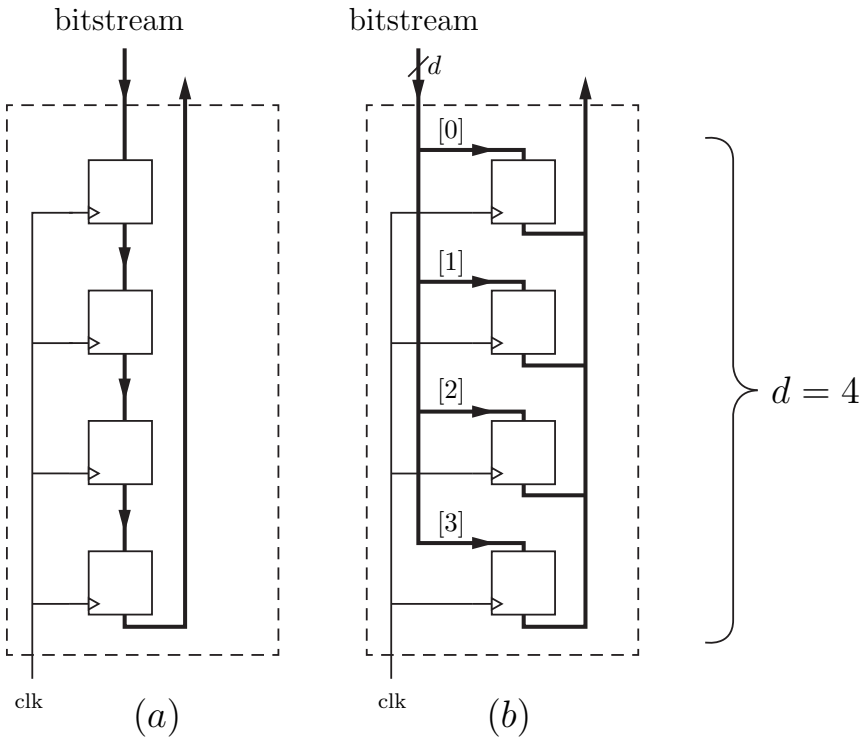


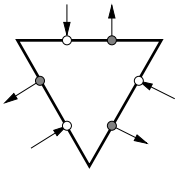
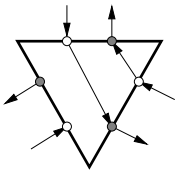
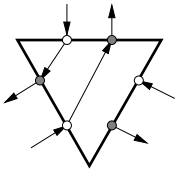
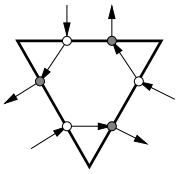
Figure 3.4: An example of the two proposed methods for implementing frames with more than one configurable bits ($d = 4$). Dashed line shows the boundary of a frame. The two methods are (a) serial method where the bitstream is a 1-bit data line and is shifted through each flip-flop of the frame, (b) parallel method where the bitstream has a width of d and configures the flip-flops all at once in 1 clock cycle.

3.2 Basic Network Control Plane

The control plane of the scan-path network too has an underlying tree of the form shown in Figure 3.1. At the leaves are the n elements of the scan set $\mathbb{S}$. The k elements of the configuration set $\mathbb{C}$ are distinguished by a flag f_i that is 1 if and only if $i \in \mathbb{C}$ meaning that frame i needs reconfiguration. Consider any internal node x in the underlying binary tree. Notice from Figure 3.1 that if none of the leaves of x are in $\mathbb{C}$, then none of the ports of x are internally connected. The other three internal configurations of interest can also be similarly categorized as shown in Table 3.1. We use the following notation in the

table. We denote by $\mathcal{T}_x, \mathcal{T}_x^\ell, \mathcal{T}_x^r$, the subtrees rooted at x, its left child and its right child, respectively. A tree $\mathcal{T}$ will be said to be *active* if and only if there is a leaf i in $\mathcal{T}$ such that $i \in \mathbb{C}$; recall that the leaves of the tree are elements of $\mathbb{S}$. The global conditions that

Table 3.1: Control plane operation; "$*$" indicates a don't care.

global condition	local condition			control signals					internal configuration
	α_x	β_x	γ_x	c_0	c_1	c_2	c_3	c_4	
$\mathcal{T}_x$ is not active	0	0	0	0	$*$	$*$	0	0	
$\mathcal{T}_x^\ell$ is not active, but $\mathcal{T}_x^r$ is	0	1	1	1	1	1	1	0	
$\mathcal{T}_x^r$ is not active, but $\mathcal{T}_x^\ell$ is	1	0	1	1	0	$*$	0	1	
Both $\mathcal{T}_x^\ell$ and $\mathcal{T}_x^r$ are active	1	1	1	1	1	0	1	1	

characterize the internal configurations of a node can be reduced to simple local conditions by propagating "indicator bits," described below.

Each leaf i sends its flag f_i as the indicator bit to its parent; recall that $f_i = 1$ if and only if $i \in \mathbb{C}$ (simply meaning that the frame needs to be reconfigured). We call a node active iff it receives a 1 indicator bit. Let each control node x receive indicator bits α_x

and β_x from its left and right children. Then it uses the internal configuration shown in Table 3.1 to generate the values of the control signals $c_0(x), c_1(x), \cdots, c_4(x)$ of Figure 3.3 and also sends indicator bit $\gamma_x = (\alpha_x \text{ OR } \beta_x)$ to its parent. In this case, it is obvious that a 1 will propagate through the control nodes' OR gates up to the node x even if $f_i = 1$ for only one of x's descendant leaves. Therefore, the local conditions of Table 3.1 reflect the global condition and the scan path will be constructed. Figure 3.5 shows the internal

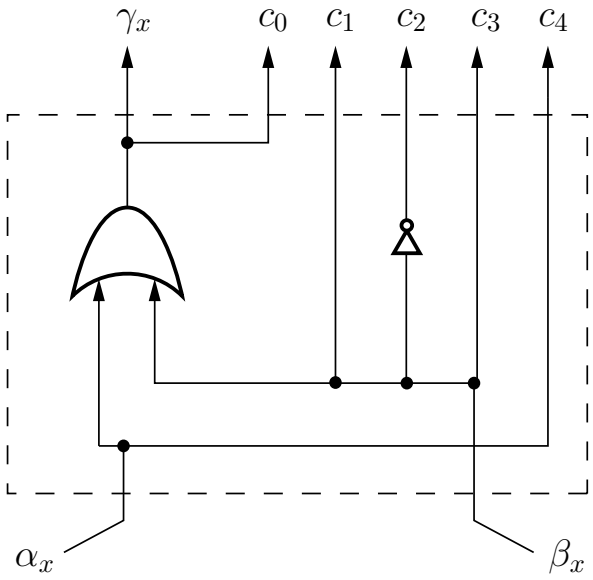


Figure 3.5: Internal structure of the control node x.

structure of the Basic Network's control node x. Table 3.1 was used as a truth table to derive the control signals $c_i(x)$ for this circuit. The values marked with "$*$" are "*don't care*" which have no effect on the output if take either value of 0 or 1 but are chosen in a way to result in a simpler circuit.

The Basic Network also has a control. This unit, as the name suggests, is a module that orchestrates the operation of the whole network. It provides an interface between different modules of an architecture and also to the outside. The term "outside" in this context can mean any other module with which the main module is interacting. In Basic Network, the controller is a part of the control plane and has three inputs *start_init*, *reset*, *clk*, and an output *done_init* (see Figure 3.6). Its main function is to generate a *done_init* when the

paths in the tree have been completely set and the network is ready for the data to be shifted. In other words, *done_init* will be set to 1 when the initialization phase is done. To achieve this function, the controller can be implemented using a Finite State Machine (FSM) that operates as follows. In its initial state, the FSM waits for a *start_init* signal (that initiates PR) to begin the operation. Upon receiving *start_init*, an internal $\log\log n$-bit counter starts counting and the FSM stays in this state until the counter counts up to $\log n$. The reason behind designing such counter is that the time it takes for the indicator bits to propagate to the root of the scan tree has a $O(\log n)$ combinational delay. That is because the longest combinational path that the indicator bits of the control plane, namely f_i flags, have to pass to reach the root of the tree from the leaves is proportional to $\log n$ OR gate delays. Therefore, the time it takes for a $\log\log n$-bit counter to count up to $\log n$ (with a carefully chosen clock to be slightly greater than an OR gate delay) is going to give a good estimate of initialization time. After the counter finishes the count, the FSM goes to its final state where it sets the output signal *done_init* to 1 indicating that the Basic Network is ready for the bitstream. The *done_init* signal is passed on to the unit responsible for dispensing the configuration bitstream. Figure 3.6 shows a block diagram of the Basic Network. Tree nodes of the data plane ("*data nodes*") are labeled as "DNi" and those of the control plane ("*control nodes*") "CNi" where i is node index. The connection between CNi and DNi consists of the five control signals $c_0(i), c_1(i), \cdots, c_4(i)$. Leaf nodes are labeled as "DFFi" (as it consists of a D flip flop).

Observe that each node of the control and data planes uses a constant amount of logic, so its cost and delay are both constants meaning they do not change with the number of frames n (which we call the size of the network). Since the number of internal nodes in the network is $n-1$, the hardware cost of the Basic Network is $\Theta(n)$. Note that the height of a binary tree with size n is $\log n$. Therefore, the control plane determines all the control inputs with $O(\log n)$ combinational delay as said earlier. The scan path also generates a path out of the last element of $\mathbb{C}$ (shown dashed in Figure 3.1), but as observed earlier, it

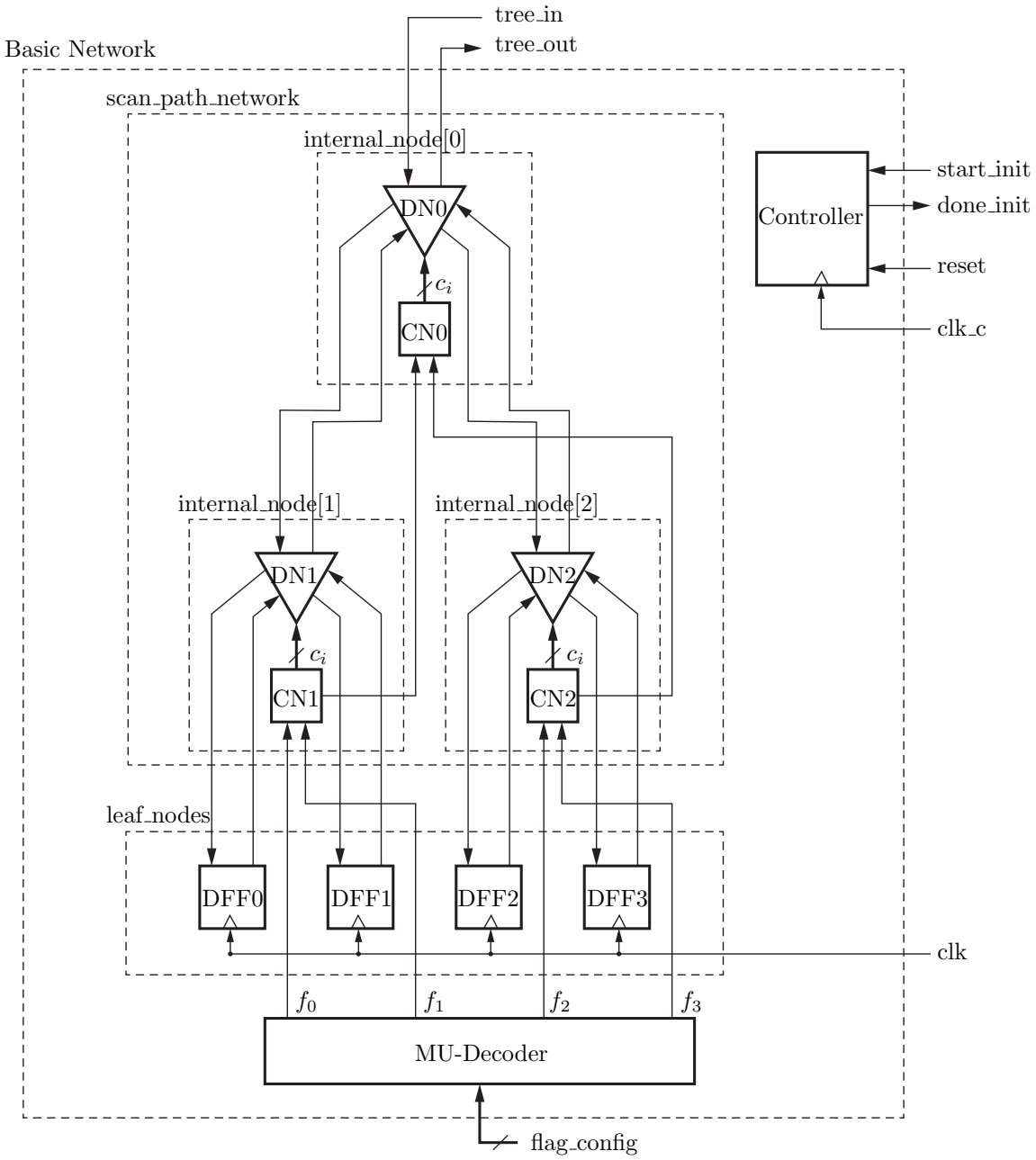


Figure 3.6: Block diagram of the Basic Network of size $n = 4$. Notice that the MU-Decoder mentioned in Chapter 1 is used at the input of the Basic Network and it allows for random selection of flag bits f_i when provided with (for example) a $\log n$-bit input *flag_config*.

is of no consequence to the speed of operation of the scan-path because this path is placed after the last sequential element of the logic; it exists only due to the uniformity in the internal node design.

Notice that the header path π_0 of the scan path has $\Theta(\log n)$ delay. Other paths π_j (for $0 < j < k$) have $O(\log n)$ delay in the worst case, but can be substantially shorter (for example, path π_2 of Figure 3.1); we also address path lengths in Chapters 4, 5, and 6. Therefore, we have the following result.

Lemma 1 For any $1 \leq k \leq n$, let scan set $\mathbb{S}$ and configuration set $\mathbb{C}$ have n and k elements, respectively. There exists a scan-path network that can generate a scan-path through the elements of $\mathbb{C}$, such that its hardware cost is $\Theta(n)$, its delay is $\Theta(\log n)$ and the scan-path clock cycle time is $\Theta(\log n)$. ■

This result is without the header-path network and clock recommendation (Chapters 5 and 6). Lemma 1, coupled with the fact that $t_h = t_c = 0$ in Equation (2.4), gives the following result.

Theorem 2 There exists a scan-path network of size $\Theta(n)$ that can configure any set of k elements out of a set of n elements in $\Theta(k \log n)$ time. ■

Remark: The above results are for the Base Network described in this chapter.

Chapter 4
The Shortcuts Network

In Chapter 3 (the Basic Network), we used a balanced binary tree as the underlying structure. This has the disadvantage of a large $O(\log n)$-delay path when two adjacent elements of $\mathbb{C}$ lie on different halves of $\mathbb{S}$, even if the indices differ by a small amount (for example, if $\frac{n}{2} - 1, \frac{n}{2} \in \mathbb{C}$, then the topological distance between them would be $2 \log n$). In this chapter, we augment the tree in the Basic Network by adding an edge between adjacent leaves (see Figure 4.1). This requires additional "leaf switches" that determine whether a path is going up the tree or directly moving right to the next leaf (using the shortcut). This would, for example, reduce the distance between $\frac{n}{2} - 1$ and $\frac{n}{2}$ to 1. In fact,

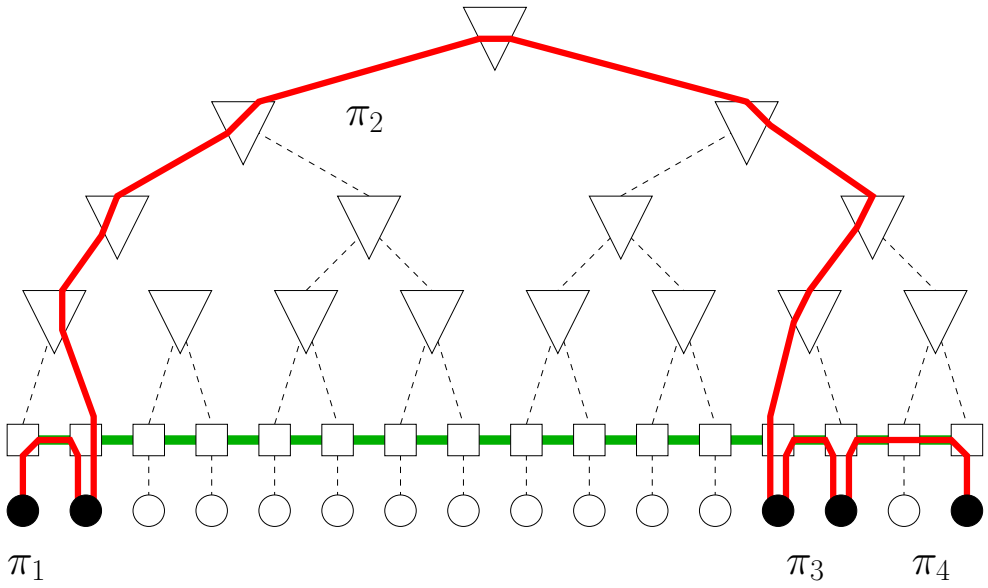


Figure 4.1: An example path in the Shortcuts Network. The underlying tree is shown dotted. The new set of switches at the leaves are shown as squares (see also Figure 4.2) and the connection between these switches is shown in green. The header path π_0 is omitted for clarity. Observe that the first two leaves and the last three leaves in the configuration set are connected by shortcut edges. The connection from leaf 2 to leaf 3 traverses the tree edges (length 10 from "circle-to-circle"); in contrast, the shortcut path between these two leaves has length 13.

for the case where $\mathbb{C}$ has contiguous elements of $\mathbb{S}$ (see Section 5.1), the scan path delay is a constant. However, for other pairs of leaves, for example $\frac{n}{3}$ and $\frac{n}{2}$, the path in the original tree is better. Note that adding a shortcut network does not change the worst case delay since there is still uncertainty about how the elements of $\mathbb{C}$ are distributed among the leaves of the tree; the worst case delay is still $2 \log n$ and consequently we will need a $O(\log n)$-cycle clock. However, having shortcuts added to the Basic Network improves the probability of a faster clock in some cases. In Chapter 5, we will introduce yet another augmentation to the Basic Network that will enable it to detect such situations in which the clock can be improved.

4.1 Shortcuts Network Data Plane

As said earlier, the Shortcuts Network is an enhancement to the Basic Network to improve its performance. The Shortcuts Network's data plane consists of n leaf switches at the base of a tree structure (Figure 4.1). Whenever appropriate, these switches enable the data path to skip the "tree path" and instead use a faster route through the "shortcut path" to reach the next active leaf. The rest of the data plane is identical to that of the Basic Network including the tree structure and the leaf nodes. Figure 4.2 shows the structure of the switch at each leaf x. The switch receives three inputs $p_i(x)$ (from its parent), $s_i(x)$ (the shortcut input from its left) and $f_i(x)$ (from the "frame" that the leaf represents). Table 4.1 shows the eight configurations of leaf switches that are required in order to satisfy all the possible states of the datapath; all except the case in Table 4.1(g) are illustrated in Figure 4.1. By appropriately setting the control signals $c_i(x)$ for $0 \leq i \leq 4$, the eight configurations of a leaf switch can be realized. The proper values of these control signals are also shown in Table 4.1. The value shown as "$*$" is a don't care value and can be chosen arbitrarily. Also, for the cases (b) and (c) (the first and the last leaves), there are two possible cases depending on the configuration set. The control signals for both cases (i.e., shown with solid and dashed lines) are written before and after a slash. Furthermore,

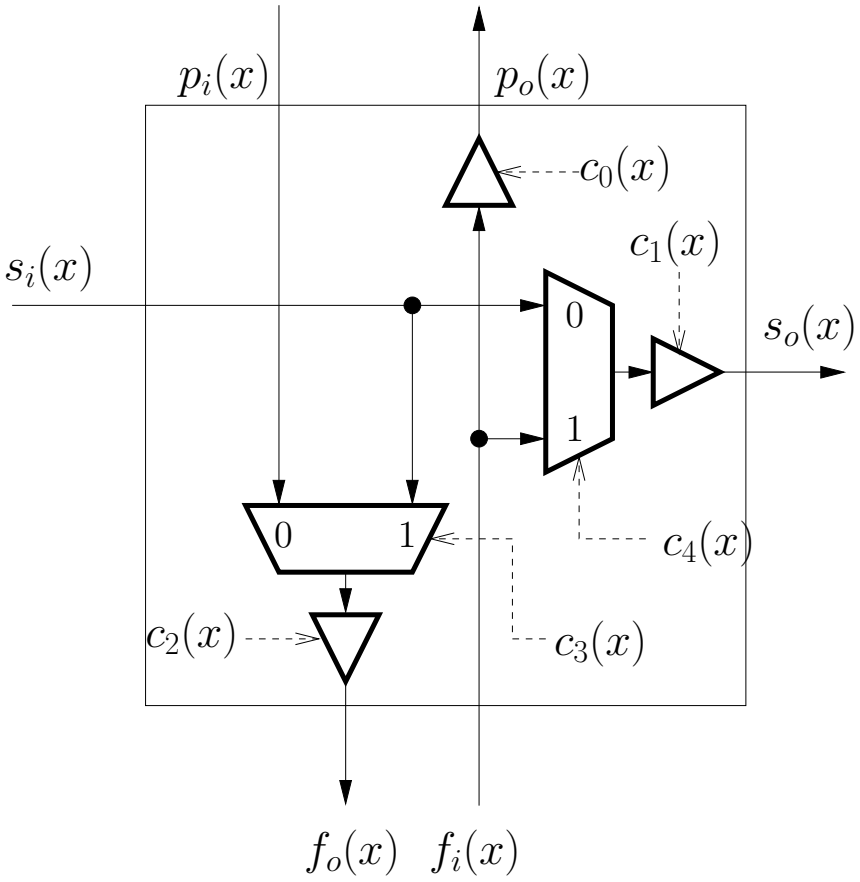


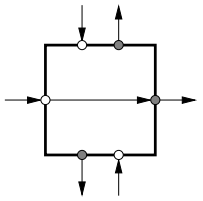
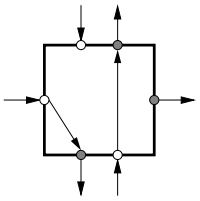
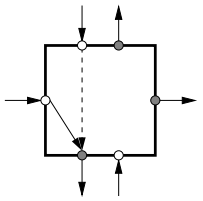
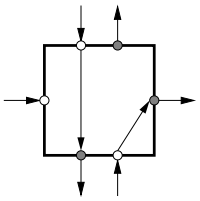
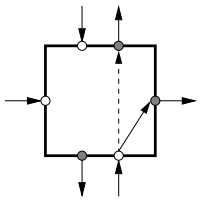
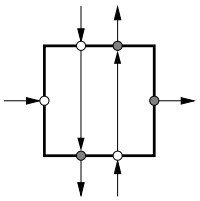
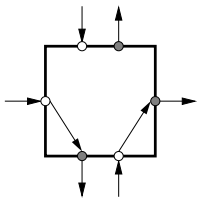
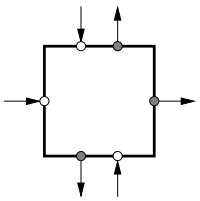
Figure 4.2: Internal structure of the data plane of a leaf switch x.

for cases (a) and (h) in which the corresponding leaf is inactive, the signal c_0 is chosen to be a don't care instead of a zero. That is because the parent of this switch is an internal node with already "disconnected" ports, therefore, choosing "$*$" for c_0 would not affect the function of the switch. In the Section 4.3, we will cover how these control signals are produced in the control plane.

4.2 Selecting the Right Path

In this section we describe a method by which a pair of selected adjacent leaves in $\mathbb{C}$ can decide which path to use, the one with tree edges or the one with the, newly added, *shortcut* edges.

Table 4.1: Configurations of a leaf switch

Configuration	Remarks / Control Signals	Configuration	Remarks/ Control Signals
 (a)	shortcut passing through $c_0 = *$, $c_1 = 1$, $c_2 = 0$, $c_3 = *$, $c_4 = 0$	 (e)	end of shortcut path, beginning of tree path $c_0 = 1$, $c_1 = 0$, $c_2 = 1$, $c_3 = 1$, $c_4 = *$
 (b)	last leaf $c_0 = *$, $c_1 = *$, $c_2 = 1$, $c_3 = 1/0$, $c_4 = *$	 (f)	end of tree path, beginning of shortcut path $c_0 = 0$, $c_1 = 1$, $c_2 = 1$, $c_3 = 0$, $c_4 = 1$
 (c)	first leaf $c_0 = 0/1$, $c_1 = 1/0$, $c_2 = *$, $c_3 = *$, $c_4 = 1/*$	 (g)	end and beginning of tree paths $c_0 = 1$, $c_1 = 0$, $c_2 = 1$, $c_3 = 0$, $c_4 = *$
 (d)	end and beginning of shortcut paths $c_0 = 0$, $c_1 = 1$, $c_2 = 1$, $c_3 = 1$, $c_4 = 1$	 (h)	no path $c_0 = *$, $c_1 = 0$, $c_2 = 0$, $c_3 = *$, $c_4 = *$

The idea is to propagate control information through both the tree and shortcut paths and select the shorter one. Let i_1, i_2 be a pair of adjacent elements of $\mathbb{C}$. As in Chapter 3, i_1 and i_2 each send an indicator bit up the tree until they meet at their lowest common ancestor. Here, we also require these indicator bits to come back down the tree. That is, the bit from i_1 reaches i_2 and vice versa. Additionally, i_1 and i_2 each also sends a different indicator bit along its shortcut edge. Assume that the control plane of each internal node has flip-flop(s) to shift information up and down the tree. Similarly, let each leaf have flip-flop(s) to shift information along the shortcut edges. It is important to note that since the flow of the bitstream (in the reconfiguration phase) is from i_1 to i_2 later, node i_1 has to decide how to set its switches (via control signals c_i in Figure 4.2) in order to send the data in the correct direction (i.e. via the shorter path); similarly, i_2 has to properly set its control signals to receive the data from the shorter path. That is why i_1 and i_2 both need to send the control information to each other through the tree and shortcut edges. Figure 4.3 illustrates this idea.

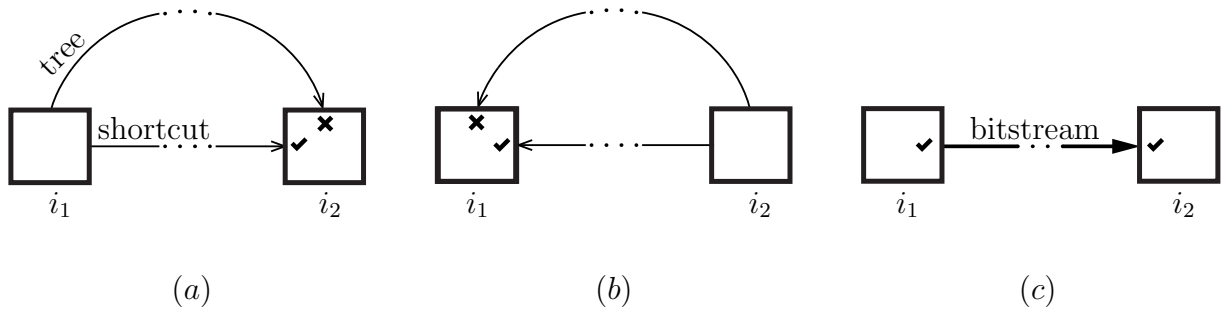


Figure 4.3: The procedure of selecting the right path between two adjacent elements of $\mathbb{C}$, i_1 and i_2; (a) i_1 sends a bit forward along both tree and shortcut paths and based on which one has arrived faster i_2 sets its *input* switches accordingly; (b) i_2 sends a bit in the opposite direction along both paths and i_1 sets its *output* switches accordingly; (c) the path in the data plane has been set and is ready for the bitstream. In this illustration the shortcut path has been shown to be shorter. The other path could also have been the shorter one in a different example. Parts (a) and (b) need not be done one after the other.

We now describe how i_1 sends information to i_2 along both paths, and how i_2 picks the shorter one; the reverse direction is similar. Let the lowest common ancestor of i_1 and i_2 be x at level ℓ. The tree path between i_1 and i_2 has length 2ℓ and the shortcut path has length $|i_2 - i_1|$. Upon receiving a "start" signal from the master controller, leaf i_1 sends a 1 to its parent and this 1 shifts along the path through the lowest common ancestor node x and then back down to i_2. Note that the internal nodes have already been set up to construct the paths in the tree using the same method as described in Section 3.2. Similarly, leaf i_1 sends another 1 through the shortcut edge to neighboring leaf $i_1 + 1$ and then to i_2. Both 1's shift through the path, one node at a time, through a sequence of flip-flops until they reach i_2 after 2ℓ and $|i_2 - i_1|$, clock cycles, respectively. Node i_2 has a finite-state machine (FSM) (see Figure 4.4) that accepts inputs from the top (tree edge) and side (shortcut edge); let these inputs be a and b, respectively. The FSM starts at an idle state where it waits for a "start" signal to be issued. Upon receiving the start signal s, the FSM goes to a wait state in which it waits for the two control signals from the top and the side to arrive. It selects one of two states "tree" or "shortcut" depending on which 1 reaches i_2 first. If both 1's reach at the same clock cycle, then the FSM can pick any one of the two states (the FSM in Figure 4.4 selects the tree path). After selecting a state, the FSM stays in that state regardless of any changes in the inputs. Additionally, there is a fourth asynchronous input *reset* which takes the FSM back to its idle state. This is indicated as the starting state in Figure 4.4. Clearly, this FSM (which is present at every leaf) has constant size and delay and does not change the asymptotic network complexity.

4.3 Shortcuts Network Control Plane

Now that we covered the idea behind selecting the right paths, we will describe the implementation of a circuit to perform this task. The process of selecting the right path takes place in the control plane of the Shortcuts Network (the data plane is still combina-

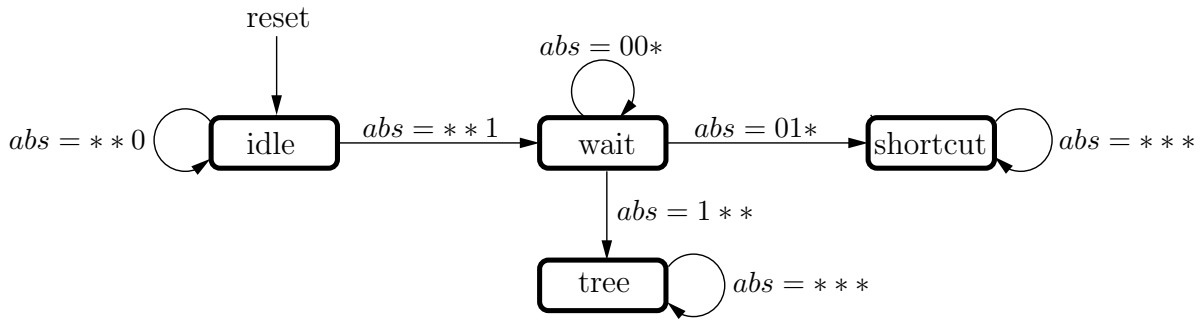


Figure 4.4: The FSM to select between the tree edges and shortcut edges. State transitions are labeled with values for the inputs a, b, and s. A don't care value is indicated by a "$*$." For example $abs = 01*$ indicates that the input can be $abs \in \{010, 011\}$.

tional.) As in the Shortcuts Network's data plane, the control plane also consists of two main components: a tree structure and a set of leaf switches.

Again, let i_1 and i_2 be a pair of adjacent elements of $\mathbb{C}$. As stated earlier, the tree structure is responsible for shifting indicator bits to give each one an estimate of the delay of that path. Therefore, the tree has to have the ability to shift the indicator bits in both directions; namely *forward* as in sending from i_1 to i_2 and *backwards* in the opposite direction. The internal structure of each node in the tree is similar to that of Figure 3.3. The only exception is that here we use flip flops at each output port in order to be able to shift the information along the tree (see Figure 4.5); in contrast Section 3.1 used a combinational circuit. Each internal node (labeled (f) as in "forward") is accompanied by a complementary circuit (marked with dashed boundary and labeled (b) as in "backward") which uses the exact same structure but in a reversed direction to enable sending the indicator bits in both directions.

As in the data plane, we have leaf switches at the base of the tree in the control plane. These switches are responsible for shifting the indicator bits in both directions and determining the shorter path by using the local FSM described in Section 4.2. Figure 4.6 illustrates the internal structure of the leaf switch x in the control plane of the Shortcuts

Network. The module labeled as (f) is responsible for shifting the indicator bits forward (to the right) through the shortcut path. It also uses the aforementioned FSM to determine the shorter path between the tree path and its corresponding shortcut path entering from the left. Like the internal control node of the tree, this module is also paired with an identical switch but in the reverse direction; namely, the switch (b) shifts the bits from the right to the left and also determines the shorter path between the incoming tree path and the shortcut path on its right side.

Forward leaf switch x^f at leaf x has two input ports "upper" (from tree) and "side" (from shortcut), $u_i^f(x)$ and $s_i^f(x)$, respectively. It also has two output ports $u_o^f(x)$ and $s_o^f(x)$ to the tree and the side, respectively. Port $u_i^f(x)$ is connected to the input of the FSM and $s_i^f(x)$ is fed to a flip-flop before entering the FSM. That is because we want to count x itself in contributing to the total delay. The FSM gets its start signal w when both *start* and $f(x)$ are asserted where *start* is the signal issued by the controller to initiate the delay assessment phase; and $f(x)$ is the flag indicating the corresponding frame is active. If $f(x) = 1$, then the leaf switch initiates the indicator bits through the output ports $u_o^f(x)$ and $s_o^f(x)$ and also determines the faster path using the FSM; otherwise, neither is performed and the "inactive" switch just shifts along the incoming indicator bits. As mentioned earlier, the function of leaf switch x^b (marked as (b) in Figure 4.6) is identical to x^f but in the opposite direction. The leaf switch at leaf x in the control plane ultimately generates two outputs g^f and g^b which along with the flag signal $f(x)$ will determine the values of the control signals $c_i(x)$ of the leaf switch x in the data plane (Figure 4.2) as follows:

$$\overline{c_0(x)} = c_1(x) = g^b(x) + \overline{f(x)}$$

$$c_2(x) = c_4(x) = f(x)$$

$$c_3(x) = g^f(x)$$

The advantage of the Shortcuts Network will become clearer after the discussion of Clock Recommendation in Chapter 5 (see Theorem 10). Additionally, preliminary analysis of the tree structure showed that adding shortcuts at other levels of the tree is not beneficial.

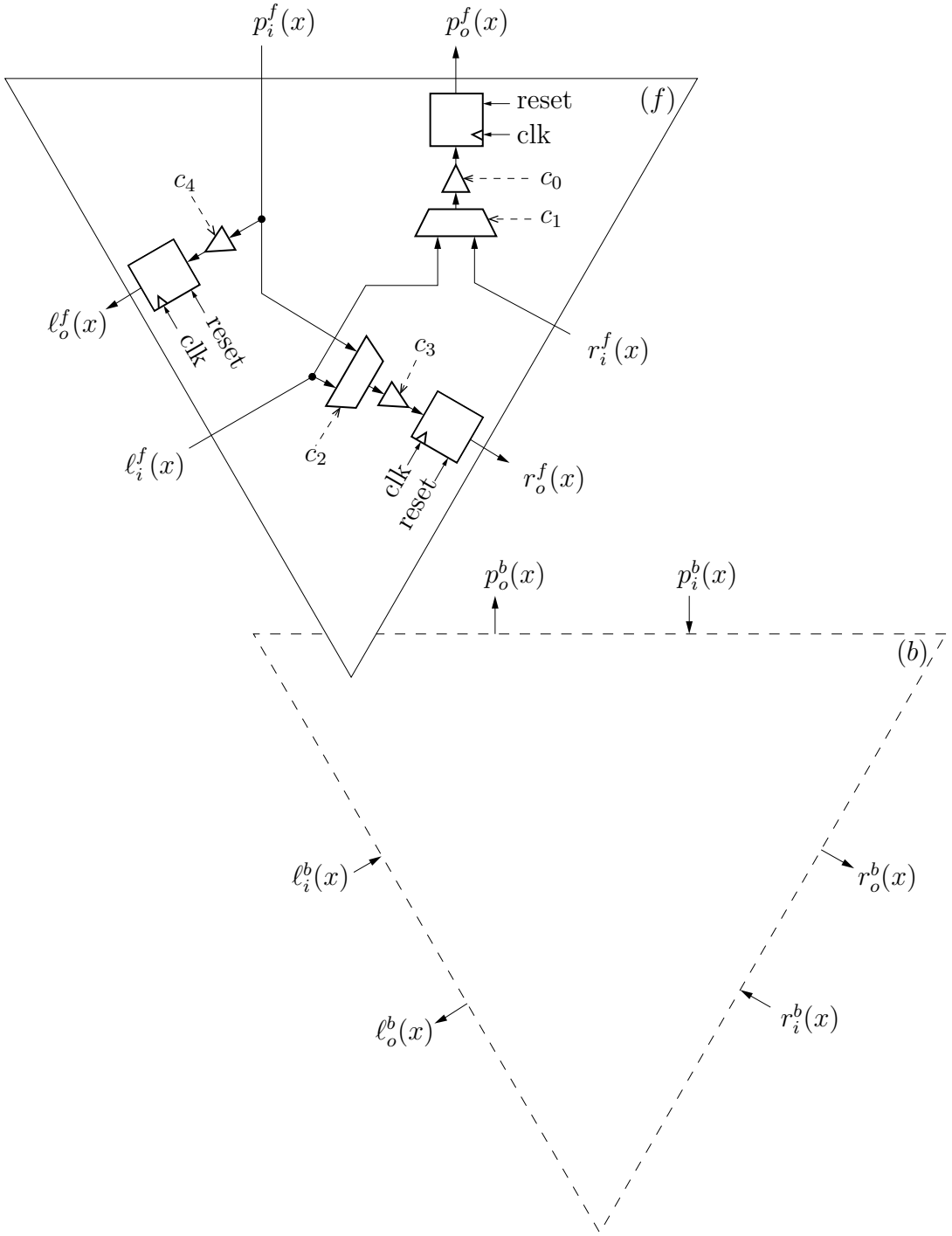


Figure 4.5: The internal structure of node x in the control plane of the Shortcuts Network. The module shown as dashed represents the internal node of a tree that shifts the indicator bits "backwards," from i_2 to i_1.

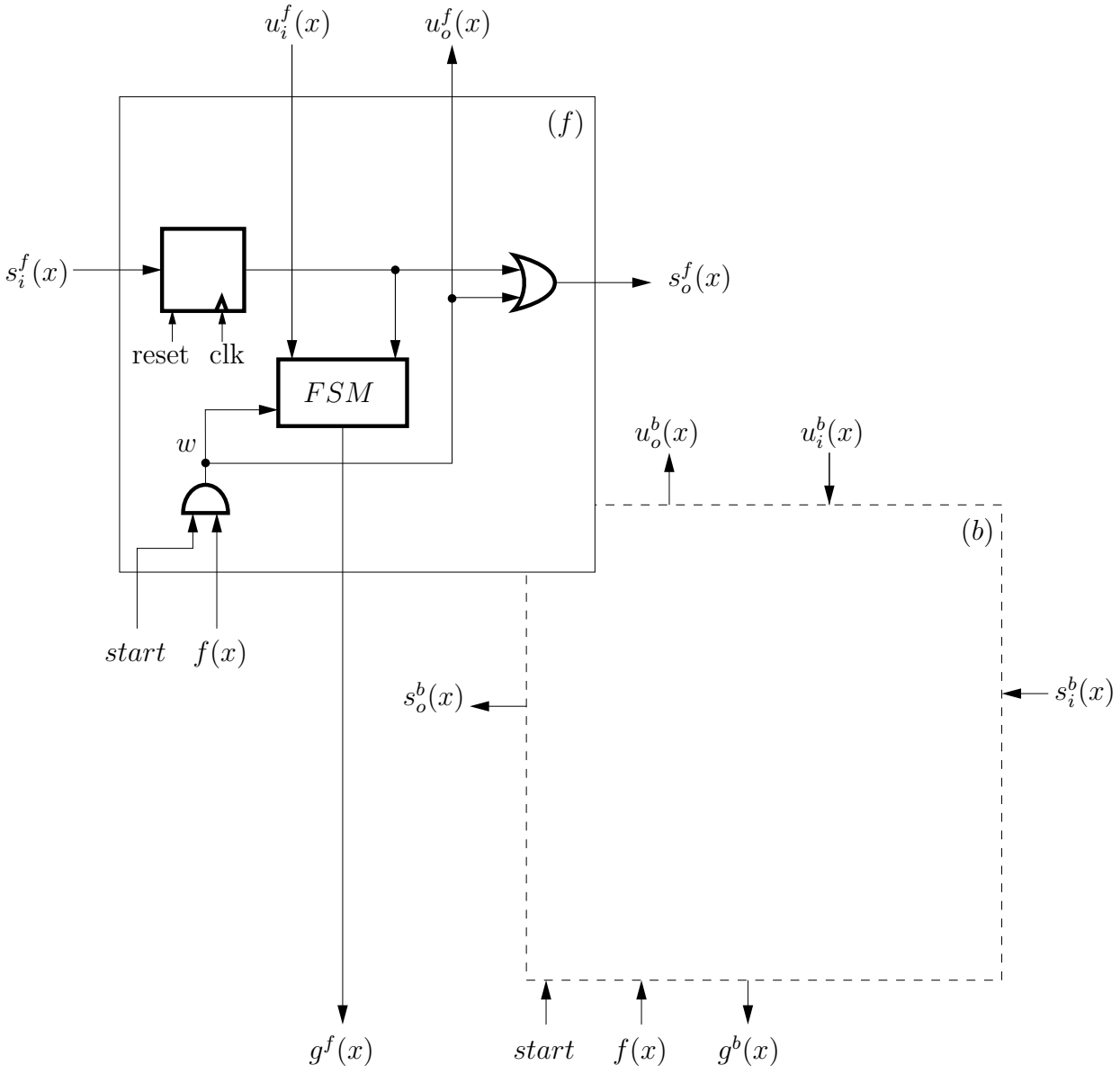


Figure 4.6: The internal structure of leaf switch x in the control plane of the Shortcuts Network. The module (b) represents the leaf switch that shifts the indicator bits "backwards" and its internal structure is the mirror image of (f). The leaf switch also generates an output $g^f(x)$ (resp. $g^b(x)$) which indicates the shorter path between the path coming down from the tree and the shortcut path entering from the left (resp. right).

Chapter 5
Clock Recommendation

Chapter 6 will effectively establish that the maximum clock rate for scanning in the bitstream is independent of the header path, π_0, depending only on the delays of the remaining paths $\pi_1, \pi_2, \cdots, \pi_{k-1}$ in the scan path network. As the example in Figure 3.1 shows, some of these paths may be very short (length of 2 for π_2) and others quite long (length $2 \log n$ for path π_3); in fact any path between adjacent nodes of $\mathbb{C}$ that are in separate halves of the tree is of length $2 \log n$ in the Basic Network which is the longest possible combinational path length in this architecture; however, in the Shortcuts Network they could be closer. The maximum clock rate is inversely proportional to the length of the longest path.

In this chapter we first identify situations where a lower clock rate can be employed by investigating two cases of selecting a configuration set from the scan set, namely, (a) "random distribution" and (b) "contiguous distribution" of the configuration set elements. Then, we will derive a network that can prescribe the correct clock rate to use in bringing in the configuration bits. We call this the "Clock Recommendation Network." Both the Basic and Shortcuts Networks are considered.

5.1 Random Distribution

There are $\binom{n}{k}$ ways to select a k-element configuration set $\mathbb{C}$ from an n-element scan set $\mathbb{S}$. Here we assume that all these ways are possible with equal probability $\frac{1}{\binom{n}{k}}$; that is, the elements of the configurable set $\mathbb{C}$ are randomly distributed over the scan set $\mathbb{S}$. For any specific choice of $\mathbb{C}$, the scan-path delay (when mapped as in Chapter 3) can range from 2 to $2 \log n$; for simplicity, and without loss of generality, we assume the delay of each path connecting two nodes (including a node on one end) to be 1. Otherwise, assuming

this delay to be a constant value D will only scale the path delay by this constant value. Therefore, assuming $D = 1$ does not affect the main principle here.

In this section we derive the average path length over all $\binom{n}{k}$ choices for $\mathbb{C}$. For any given $2 \leq k \leq n$, let the average maximum path length be $A(n,k)$. If a particular $\mathbb{C}$ includes elements from both halves of $\mathbb{S}$, then the scan-path delay is $2\log n$ (as this path must traverse the root of the tree). Let $\sigma_{n,k}$ be the probability that $\mathbb{C}$ spans both halves of $\mathbb{S}$. Then

$$A(n,k) = \underbrace{\sigma_{n,k}\,(2\log n)}_{\text{spans two halves}} + \underbrace{(1-\sigma_{n,k}) \cdot A\Big(\frac{n}{2},k\Big)}_{\text{restricted to } \frac{n}{2} \text{ leaf tree}} \tag{5.1}$$

If $k > \frac{n}{2}$, then clearly $\sigma_{n,k} = 1$ and $A(n,k) = 2\log n$. Suppose that $1 < k \leq \frac{n}{2}$. The probability that $\mathbb{C}$ is restricted to one of the halves of $\mathbb{S}$ is

$$1 - \sigma_{n,k} = 2\frac{\binom{n/2}{k}}{\binom{n}{k}} = 2\prod_{j=0}^{k-1}\left(\frac{n-2j}{2n-2j}\right) \leq \frac{1}{2^{k-1}};$$

the last inequality follows from the fact that for any $x \geq 2y \geq 0$, $\quad \dfrac{x-2y}{2x-2y} \leq \dfrac{1}{2}$. That is,

$$\sigma_{n,k} \geq \left(1 - \frac{1}{2^{k-1}}\right) \geq \frac{1}{2}.$$

From this and Equation (5.1) we have

$$A(n,k) \geq \sigma_{n,k}\,(2\log n) \geq \log n.$$

Since the worst case delay is $2\log n$, we have the following result.

Lemma 3 The average scan-path delay on the Basic Network for a random distribution of configurable elements over an n-element scan set is $\Theta(\log n)$. ■

If a Shortcuts Network is used then for a random distribution, the average shortcuts path between two elements of $\mathbb{C}$ is $\frac{n}{k}$. Thus, we have the following lemma.

Lemma 4 The average scan-path delay on the Shortcuts Network for a random distribution of configurable elements over an n-element scan set is $\Omega\left(\min\left(\log n, \frac{n}{k}\right)\right)$. ∎

Comment: Since k is typically quite small,

$$\min\left(\log n, \frac{n}{k}\right) = \Theta\left(\log n\right).$$

5.2 Contiguous Distribution

Lemma 3 shows that in the random case, there is little hope for significantly reducing the scan-path delay. However in most applications, the configurable elements are placed close to each other due to *spatial locality* of frames to be reconfigured. For example if a partial reconfiguration module spans multiple frames, then these frames would be contiguous. In this section we consider the case where all k elements of $\mathbb{C}$ are contiguous elements of $\mathbb{S}$. Here, there are only $n-k+1$ possibilities with only $k-1$ of these spanning both halves of $\mathbb{S}$. Figure 5.1 demonstrates the possible positions for the distribution of k contiguous elements over the n elements of set $\mathbb{S}$.

Therefore for $k \leq \frac{n}{2}$ and with $\alpha = k-1$, here we have $\sigma_{n,k} = \frac{k-1}{n-k+1} = \frac{\alpha}{n-\alpha}$ and $1-\sigma_{n,k} = \frac{n-2\alpha}{n-\alpha}$. From this and Equation (5.1) we have

$$\begin{aligned} A(n,k) &= 2\left(\frac{\alpha}{n-\alpha}\right)\log n + \left(\frac{n-2\alpha}{n-\alpha}\right)\cdot A\Big(\frac{n}{2},k\Big) \qquad (5.2)\\ &= 2\left(\frac{\alpha}{n-\alpha}\right)\log n + \left(\frac{n-2\alpha}{n-\alpha}\right)\cdot\left[2\left(\frac{\alpha}{\frac{n}{2}-\alpha}\right)\log\frac{n}{2} + \left(\frac{\frac{n}{2}-2\alpha}{\frac{n}{2}-\alpha}\right)\cdot A\Big(\frac{n}{4},k\Big)\right]\\ &= 2^1\left(\frac{\alpha}{n-\alpha}\right)\log n + 2^2\left(\frac{\alpha}{n-\alpha}\right)\log\frac{n}{2} + \left(\frac{n-2^2\alpha}{n-\alpha}\right)\cdot A\Big(\frac{n}{4},k\Big)\\ &= \sum_{j=0}^{u}\left(\frac{2^{j+1}\alpha}{n-\alpha}\right)\log\frac{n}{2^j} + \left(\frac{n-2^{u+1}\alpha}{n-\alpha}\right)\cdot A\Big(\frac{n}{2^{u+1}},k\Big). \end{aligned}$$

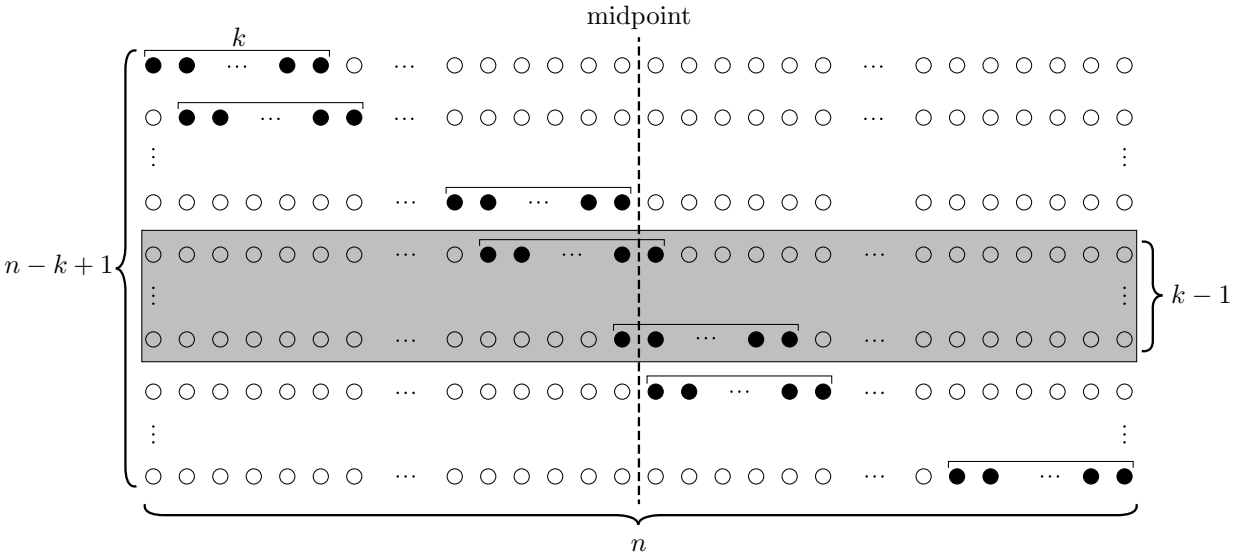


Figure 5.1: An illustration of the possible positions of k contiguous elements. The only $k-1$ cases that span both halves of $\mathbb{S}$ are enclosed in a gray box.

Let $2^u \leq \dfrac{n}{k} < 2^{u+1}$. Then $A\left(\frac{n}{2^{u+1}}, k\right) \leq 2\log k$. Now Equation (5.2) can be written as the following for $n \geq 4$.

$$\begin{aligned} A(n,k) &\leq \sum_{j=0}^{u} \left(\frac{2^{j+1}\alpha}{n-\alpha}\right)[(\log n) - j] + 2\log k \\ &\leq \left(\frac{\alpha}{n-\alpha}\right)\left[2^{u+2}\left(\log n - u + 1\right)\right] + 2\log k \end{aligned}$$

For $\sqrt{n} \geq k \geq 1$, it is easy to show that $\dfrac{\alpha}{n-\alpha} = \dfrac{k-1}{n-k+1} \leq \dfrac{k}{n}$. Additionally, since $u \leq \log n - \log k < u+1$ we have

$$\begin{aligned} \left(\frac{\alpha}{n-\alpha}\right)\left[2^{u+2}(\log n - u + 1)\right] &\leq \frac{4k}{n}\left[2^{u}(\log n - u + 1)\right] \\ &\leq \frac{4k}{n}\left[\frac{n}{k}(\log n - u + 1)\right] \\ &\leq \frac{4k}{n}\left[\frac{n}{k}(\log k + u + 1 - u + 1)\right] \\ &\leq 4\log k + 8. \end{aligned}$$

Thus,

$$A(n,k) \leq 6\log k + 8.$$

So far, we have assumed that $k \leq \sqrt{n}$. If $k > \sqrt{n}$, then $A(k,n) \leq 2\log n \leq 4\log k$. Thus we have the following result.

Lemma 5 The average scan-path delay for a contiguous distribution of k configurable elements over an n-element scan set in the Basic Network is $O(\log k)$. ■

Lemma 6 The average scan-path delay for a contiguous distribution of k configurable elements over an n-element scan set in the Shortcuts Network is constant. ■

5.3 Clock Recommendation

Lemmas 3 through 6 illustrate extreme cases for the distribution of the configuration elements. Many practical cases would lie in between, possibly closer to the contiguous distribution case as argued earlier. Therefore, there is a good chance that a fast clock could be used for many cases. This requires the module generating the clock to know the acceptable clock speed for the scan-path. In this section, we derive a circuit that returns a measure of the delay of the scan path; this can be used to select an appropriate clock rate for scanning in the configuration bitstream.

Recall that the scan path consists of k flip-flops with combinational paths $\pi_0, \pi_1, \cdots, \pi_{k-1}$ to their inputs. Let t_i denote the delay of path π_i. As will be shown in Chapter 6, t_0 can be assumed to be 1. Therefore, the scan-path delay is $t_s = \max\{t_i \ : \ 1 \leq i < k\}$. It is easy to observe from Figure 3.1 on page 17 that if the highest level that path π_i reaches is ℓ_{max}, then its delay is $2\ell_{max}$; here ℓ_{max} is the maximum height of a lowest common ancestor. Thus by detecting a node x of the tree where the path connecting two adjacent elements of the configuration set $\mathbb{C}$ turns from the left child to the right child, one can ascertain the level of x, and hence, the delay of that path. Now, if a circuit can find the highest level at which

exists a node with such property exists, then that would indicate the maximum admissible clock rate. Additionally, even though this level can be as large as $\log n$ (a $\log \log n$-bit quantity), each node should remain of constant size so that the entire network can be of size $\Theta(n)$. Now we will describe such a network that we call the Clock Recommendation Network.

Again, we use a tree structure similar to that of the Basic Network (Figure 3.1 on page 17). Each node x first sets a flag ρ_x to 1 if and only if the path turns from its left child to its right child; all this requires is setting $\rho_x = (\alpha_x \text{ AND } \beta_x)$ (see Table 3.1). For subsequent discussion we will term these nodes as *turn nodes*. Care must also be taken to ignore the dummy last path π_k.

Next at clock cycle $t \geq 0$, each node x, with left and right children y and z, respectively, generates a bit ξ_x^t as follows:

$$\xi_x^0 = \rho_x \quad \text{and} \quad \text{for } t > 0, \;\; \xi_x^t = \left(\xi_x^{t-1} \text{ OR } \xi_y^{t-1} \text{ OR } \xi_z^{t-1}\right).$$

Observe that if $\xi_x^t = 1$, then for all $t' > t$, we have $\xi_x^{t'} = 1$.

Figure 5.2 shows the internal structure of node x. Note that at $t \leq 0$ before the clock assessment phase begins $start = 0$, therefore, the D flip flop stores the value of ρ_x on its output. When signal *start* hits ($t > 0$), the circuit changes its mode to generate the output ξ_x^t as expected. Also, it is important to note that for every node located at the first level above the leaves, $\xi_y^{t-1} = 0$ and $\xi_z^{t-1} = 0$ for every t. Since these signals are input to a 3-input OR gate, having this gate is unnecessary for the nodes at $level(x) = 1$ and the output of the flip flop can be directly fed to the Multiplexer.

For any node x, let $Desc(x)$ be the set of descendants of x (including x itself). Let $Turn(x) = \{y \in Desc(x) \;:\; \rho_y = 1\}$ be the subset of descendants of x that are turn nodes. For any node x, let $level(x)$ be its level. Let the *turning level*, $L(x) = \max\{level(y) \;:\; y \in$

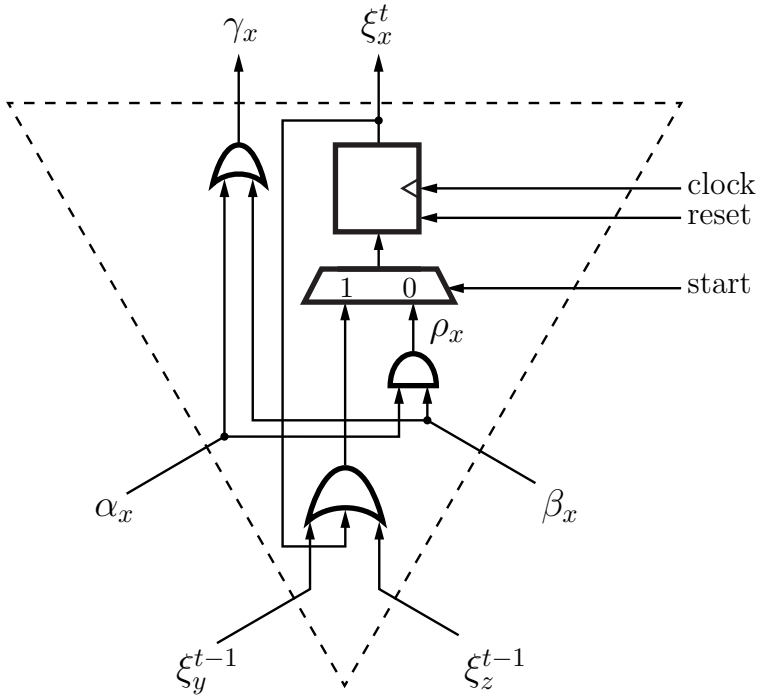


Figure 5.2: The internal structure of node x in the Clock Recommendation Network.. Signals *start* and *reset* are the outputs of the controller unit which will be covered later in this chapter.

$Turn(x)\}$, of node x be the level of the highest turning descendant of x. The following lemma is easy to prove by induction on the level ℓ of the nodes.

Lemma 7 For any node x with turning level $L(x)$, bit $\xi_x^t = 1$ iff $t \geq level(x) - L(x)$. ■

For the root r of the tree, $L(r) = \ell_{max}$ is the level reached by the highest delay path. The sequence of bits produced by the root (ξ_{root}^t) over $\log n$ steps is

$$C = \underbrace{0, 0, \cdots, 0}_{\log n - \ell_{max}} \; \underbrace{1, 1, \cdots, 1.}_{\ell_{max}}$$

zeroes ones

Observe that C is a unary representation of ℓ_{max}. A separate $O(\log n)$-state finite state machine (FSM) can accept a $\log n$-bit sequential input C from the root and convert this to the

corresponding $\log\log n$-bit binary number; this FSM is just a special purpose $O(\log\log n)$-bit counter with C as its enable or clock. The $\log\log n$-bit output is a measure of the delay of the scan-path and can be used to derive the maximum clock rate for scanning in the configuration bitstream. Figure 5.3 shows the overall architecture of the Clock Recommendation Network for $n = 8$. The flags f_i entering the tree are the same as those in Figure 3.6 and indicate the active nodes. The sequence of bits produced by the root enters the Clock Assessment module which is a special purpose FSM.

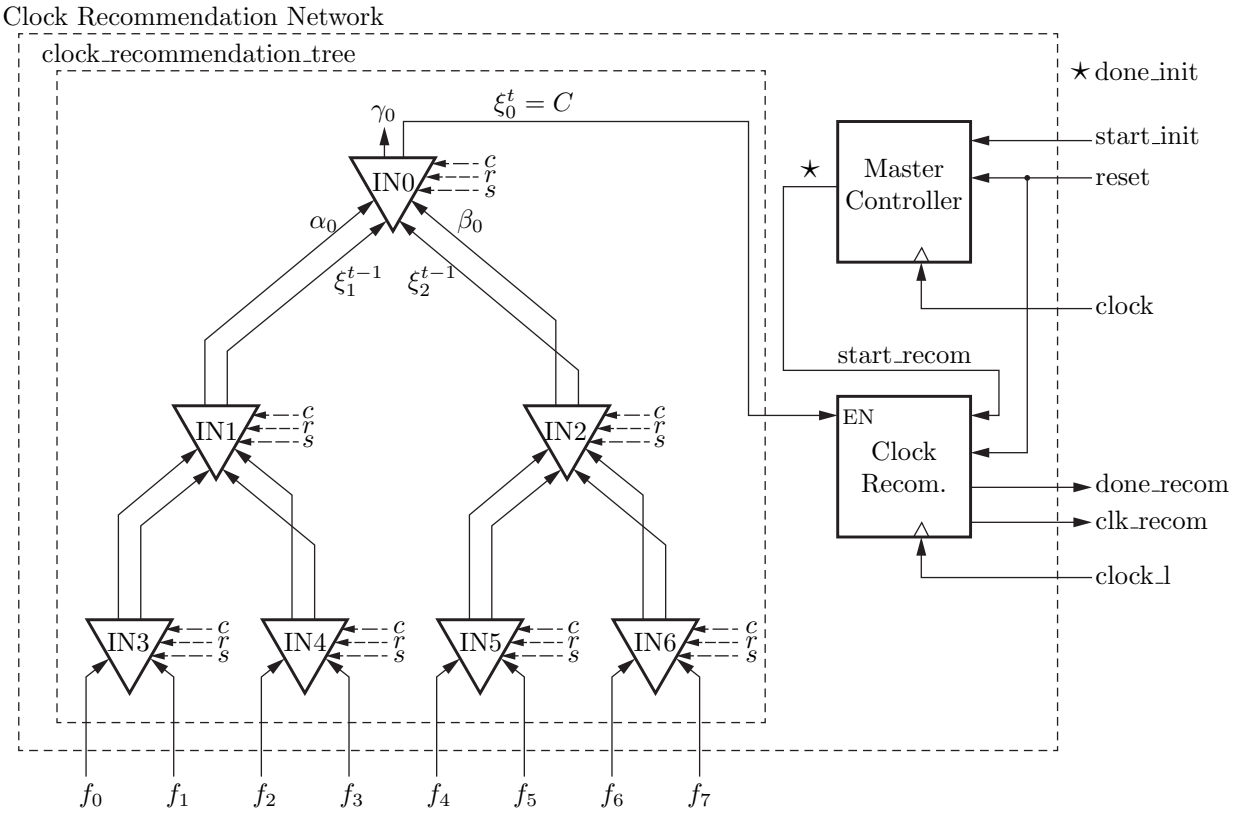


Figure 5.3: Block diagram of the Clock Recommendation Network of size $n = 8$. Internal nodes are shown as INi. Signals shown as dashed with the same labels are all feeding from the same source. Those labeled as c and r are connected to the global *clock* and *reset* respectively; and those labeled as s are connected to the Master Controller's output *done*. The output of the Master Controller *done_init* (marked with star) indicates that the nodes in the Basic Network are all set and the Clock Recommendation unit should start its operation.

5.4 Clock Generation with Shortcuts

The clock generation network for the Basic Network tree can be modified as follows to suit the Shortcuts Network (Chapter 4). First, the selected tree paths are processed as in Section 5.3; that is, merged and pipelined out through the root of the tree. Next, the shortcut paths are processed in a similar way (as explained below). The controller decides on the faster of the two. To output the shortcuts clock we proceed as follows. All leaves in a shortcut path are flagged with a 1 (initialization) and leaves that are not on a shortcut path are initialized to 0. Since the shortcuts have already been determined (see Section 4.2 on page 28) we can readily flag them. Note that since we have selected the shorter among tree and shortcut paths, a shortcut path cannot be longer than $2\log n$; thus, $2\log n$ time suffices for the process described here. The bit in each path (say from its beginning of the path) is shifted up the tree with paths merged as before (see Section 5.3). After $2\log n$ steps all bits of shortcut paths have departed from the leaves. After an additional $\log n$ steps, the sequence of at most $2\log n$ bits representing the longest shortcut path has been output through the root of the tree. This can be encoded as before using an $O(\log\log n)$ bit counter. The maximum of the lengths of the tree and shortcut paths determines the clock speed. A separate global FSM sequences the $\Theta(\log n)$ steps needed for clock generation.

The $\log\log n$-bit counter can be built with a clock cycle proportional to $\log\log\log n$. The cost of the FSM is clearly $O(n)$. Thus we have the following result.

Lemma 8 For any n, there exists a $O(\log n \log\log\log n)$-delay, $O(n)$ cost network that outputs the scan-path delay for any configurable set of an n-element scan set. ■

Comment: This lemma holds for both the Basic and Shortcuts Networks.

From Equation (2.4) on page 15, Lemmas 1 (page 25) and 8 (page 45), we have the following theorems.

Theorem 9 There exists a scan-path network of size $\Theta(n)$ that can configure any set of k elements out of a set of n elements in $\Theta(\log n \log\log\log n + kT_0)$ time, where T_0 is the

minimum clock cycle needed for the scan-path network. For a contiguous distribution of the k elements of the configuration set, this network uses an average value of $T_0 = O(\log k)$. ■

Remark: If the proposed techniques are used to reduce frame sized for a more focused reconfiguration, then n could be in the order of millions and k in the order of hundreds. Thus k can be expected to be larger than $\log n \log\log\log n$. Thus the time given by Theorem 9 is of the order of kT_0, the ideal number of cycles to input the bitstream on the proposed network.

Theorem 10 There exists a scan-path network with shortcuts that is of size $\Theta(n)$ and which can configure any set of k elements out of a set of n elements in $\Theta(\log n \log\log\log n + kT_0)$ time, where T_0 is the minimum clock cycle needed for the scan-path network. For any contiguous distribution of the k elements of the configuration set, this network uses a constant value of T_0. ■

Remark: In Theorems 9 and 10, the delay of header path π_0 is omitted. As stated earlier, π_0 imposes a $O(\log n)$ delay and the network proposed in Chapter 6 (the Header Network) addresses the problem of eliminating such a bottleneck delay.

Chapter 6
Header Network

In this context, the *header path* is defined as a path that starts at the input of the configuration bitstream and ends at the first element of the configuration set $\mathbb{C}$ (set of k selected frames out of n total frames.) Recall that in the Basic Network's data plane, the header-path π_0 is always of delay $\Theta(\log n)$ (see Figure 3.1). That is because of the fact that the number of levels from root to the leaves of an n-leaf binary tree is $\log n$, causing the header path to have the aforementioned combinational delay. Consequently, the delay of the scan-path cannot be improved (even if other paths have a smaller delay) based on Equation 2.3 from Chapter 2. In this section, we outline a method for reducing the delay of the header path π_0 to enable delays less than $\log n$.

The approach is to (1) determine the first element of configuration set $\mathbb{C}$ (the *header leaf*), and then (2) use a fast network to broadcast the configuration bitstream to all leaves, with the understanding that only the header leaf will accept it.

6.1 Determining the Header Leaf

Let $\vec{F} = \langle f_i \ : \ 0 \leq i < n \rangle$, where $f_i = 1$ if and only if $i \in \mathbb{C}$; this is the characteristic vector representing the configuration subset $\mathbb{C} \subseteq \mathbb{S}$. Suppose that $\vec{G} = \langle g_i \ : \ 0 \leq i < n \rangle$ is the prefix OR vector of $\vec{F}$, where $g_0 = f_0$ and for $i > 0$, $g_i = (f_i \text{ OR } g_{i-1})$. Consider Figure 6.1 which shows a simple implementation of a prefix OR by cascading n OR gates. Notice that any arbitrary picked output can be written as

$$g_i = \sum_{i=0}^{i} f_i.$$

It is easy to see that if i_0 is the header leaf, then $g_i = 1$ if and only if $i_0 \leq i < k$. Therefore, the prefix OR vector of $\vec{F}$ will result in $\vec{G} = \{00\ldots00111\ldots11\}$ where the

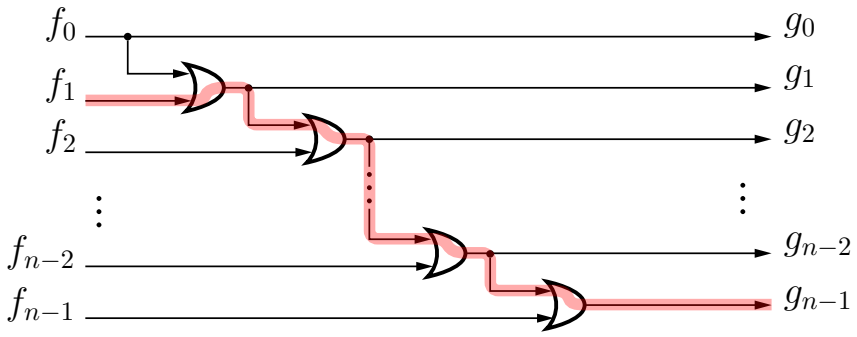


Figure 6.1: An example illustrating prefix OR of n bits by cascading $n-1$ OR gates. The worst case delay is $n-1$ OR gate delays (critical path is highlighted).

first occurring 1 is g_{i_0}. Notice that in this circuit, the worst case delay path starts at f_0 and ends at g_{n-1} and has a combinational delay of $n-1$ OR gate delays which is considered very slow.

Now, let $h_0 = g_0$ and for $i > 0$, $h_i = (g_i$ EX-OR $g_{i-1})$. Clearly given $\vec{G}$, h_i can be determined with $n-1$ EX-OR gates and one EX-OR gate delay. Figure 6.2 shows the circuit of this function. It is obvious that $h_i = 1$ if and only if $i = i_0$. In other words, we will have $\vec{H} = \langle h_i \ : \ 0 \leq i < n \rangle = \{00\ldots00100\ldots00\}$ where the header leaf is the only bit that is 1. As stated earlier, the delay of the circuit shown in Figure 6.1 is not acceptable

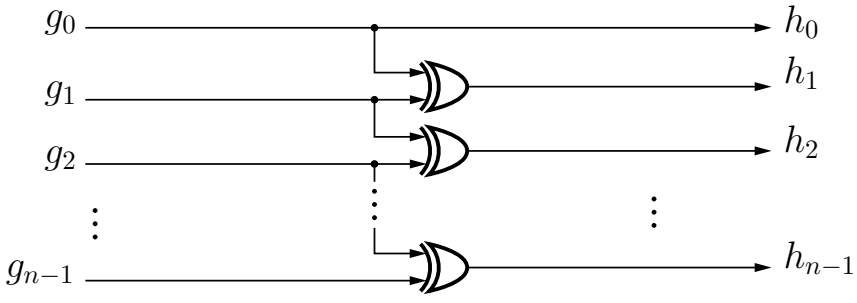


Figure 6.2: An illustration of EX-OR of n bits.

for our design since we desire to improve the delay of the header path from $O(\log n)$. As a result, a better approach is needed to determine $\vec{G}$, given $\vec{F}$. This follows a well known approach for solving recurrences [37] that is illustrated in Figure 6.3 for our context.

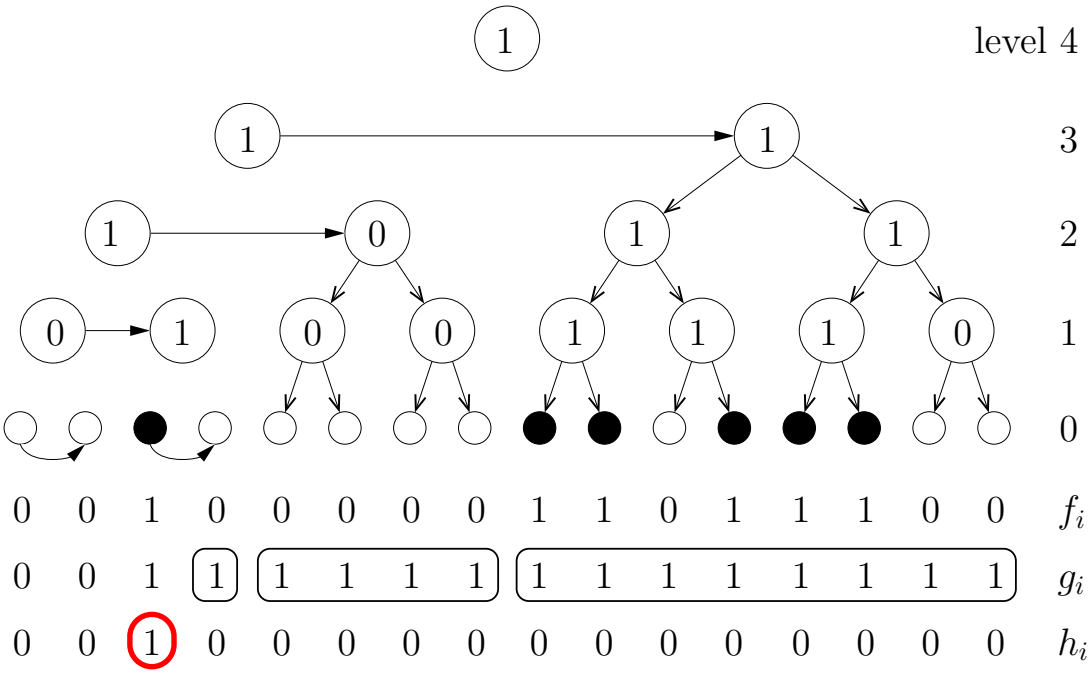


Figure 6.3: An example illustrating prefix OR of n bits using a binary tree.

Consider a binary tree with n leaves at the bottom holding the values of $\vec{F}$. The computation proceeds in two phases on the tree. In the first phase, leaf i assumes the value f_i and each internal node x holds the logical OR γ_x (say) of the values of all leaves in its subtree. The bits within circles in Figure 6.3 show these values for an example; we have used a slightly different example than the ones used before to illustrate the procedure better. This computation is identical to that used to generate the γ_x values and establish the scan path (see Table 3.1).

The second phase proceeds from the root down to the leaves in $\log n$ steps (one per level) starting from the children of the root (level $\log n - 1$). Let nodes x and y be the left and right children of the root, respectively. In the first step of the second phase, x sends γ_x to y. If $\gamma_x = 1$, then y instructs all leaves i of its subtree to set $g_i = 1$; otherwise, nodes at level $\log n - 2$ or lower proceed recursively as described below. Consider a node x at level $\log n - t$ (where $1 < t \leq \log n$) that is the left child of its parent z. Let y be the right child of z (that is, the sibling of x). If z has not instructed its leaves to set $g_i = 1$, then at

step t, nodes x and y proceed recursively as described earlier; namely, node x sends γ_x to y, then if $\gamma_x = 1$, then y instructs its leaves to set $g_i = 1$; otherwise level $t-1$ proceeds recursively. The value generated at the leaves after any sibling communication is g_i. The recursion terminates at the leaves.

Figure 6.3 illustrates the second phase. In level 3, the left child of the root sends a 1 to the right child, which instructs all its leaves to set $g_i = 1$. (This is indicated in the figure as tree edges, as opposed to an edge from a left child to its sibling. All leaves i for which we set $g_i = 1$ due to this step are enclosed in a box.) For level 2, the leftmost node similarly sends a 1 to its sibling which too causes all its leaves to set $g_i = 1$. At level 1, the leftmost node sends a 0 to its sibling, and both their descendants proceed recursively. For the first pair of leaves $\gamma_0 = \gamma_1 = 0$; leaf node 0 sets $g_0 = \gamma_0 = 0$ and sends a $\gamma_0 = 0$ to leaf node 1, which sets $g_1 = \gamma_1 = 0$. The second pair of leaves have $\gamma_2 = 1$ and $\gamma_3 = 0$. Here leaf node 2 sets $g_2 = \gamma_2 = 1$ and sends $\gamma_2 = 1$ to leaf node 3. Because the received bit $\gamma_2 = 1$, leaf node 3 sets $g_1 = 1$ (even though $\gamma_3 = 0$). A formal proof of correctness of this method in a more generalized form appears in Dharmasena and Vaidyanathan [37].

We observe that while the algorithm for Phase 2 is cast recursively, it can be easily unfolded into a circuit with $\Theta(\log n)$ delay and $\Theta(n)$ gates as outlined below. At the end of Phase 1, each tree node x holds γ_x, the OR of the bits at the leaves of the subtree rooted at x; Figure 6.3 shows these values within the circles representing nodes. Recall (see Table 3.1) that node x receives α_x and β_x from its left and right children and determines $\gamma_x = \alpha_x$ OR β_x. Phase 2 proceeds from the root down to the leaves and node x receives an input δ_x from its parent; in addition, it has input α_x received during Phase 1. Node x produces outputs ζ_x^ℓ, ζ_x^r to its left and right child, respectively. In this notation, if the left and right child of x are u and v, respectively, then $\zeta_x^\ell = \delta_u$ and $\zeta_x^r = \delta_v$. For the root,

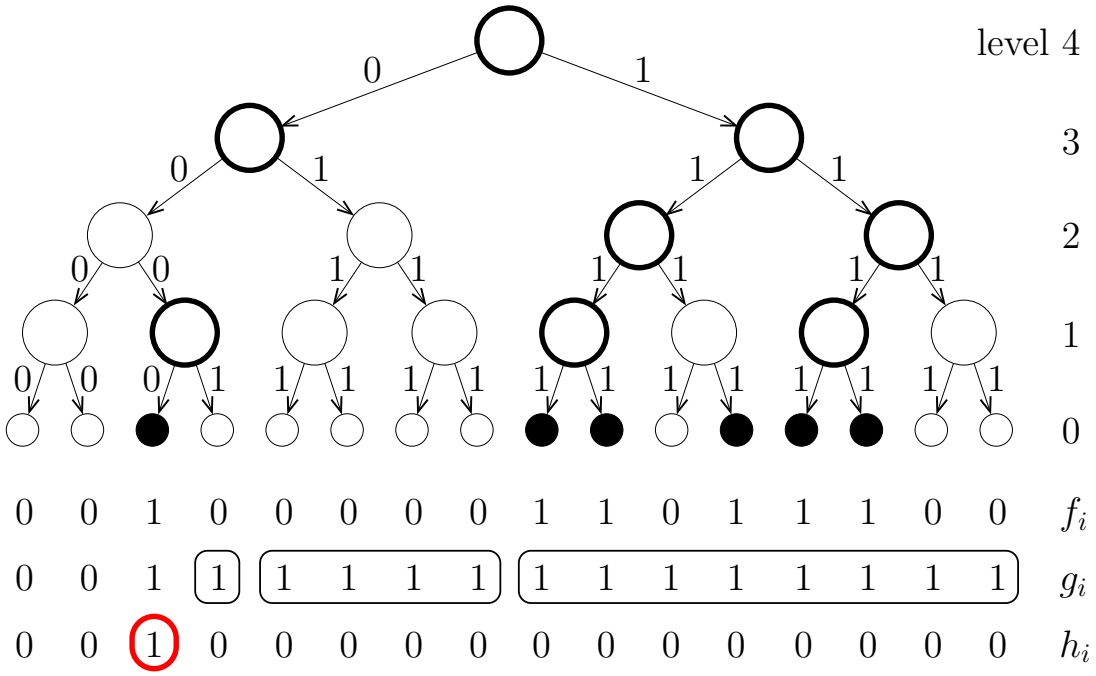


Figure 6.4: An example illustrating the non-recursive implementation of a prefix OR of n bits. A node x drawn in bold is one with $\alpha_x = 1$. The bits on an edge from a node x to its left and right children are the values of ζ_x^ℓ and ζ_x^r, respectively. Given the values of g_i, the circuit of Figure 6.2 is used to produce the h_i values that gives the header leaf (marked with red circle in the last row).

assume that $\delta_{root} = 0$. The outputs of node x are assigned as follows:

$$\begin{aligned} \zeta_x^\ell &= \delta_x \\ \zeta_x^r &= \alpha_x \text{ OR } \delta_x. \end{aligned}$$

Let leaf i receive input η_i from its parent j; here $\eta_i = \zeta_j^\ell$ OR ζ_j^r. Finally, leaf i computes $g_i = f_i$ OR η_i. Figure 6.4 illustrates these ideas. The internal structure of the internal (tree) node x and leaf i of the prefix OR tree are shown in Figure 6.5.

Finally, by putting together the prefix OR tree and the EX-OR circuit shown in Figure 6.2, the Header Network can be constructed. Figure 6.6 shows the block diagram of a simple Header Network with $n = 4$.

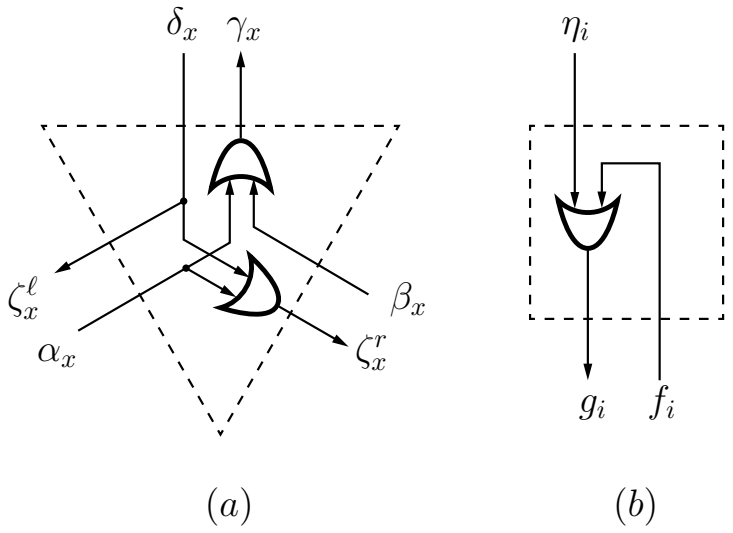


Figure 6.5: Internal structure of (a) node x and (b) leaf i in the prefix OR tree.

6.2 Constructing the Header Path

It was noted earlier that once we have determined the header leaf i (indicated by flag $h_i = 1$) the bitstream can be broadcast to all leaves with the understanding that leaf i will act on it if and only if $h_i = 1$. Such a "broadcast network" would be no more complex than the clock distribution network used to clock the chain of shift registers spanning the scan path. Thus, if the header leaf can be determined, then the header path π_0 can be constructed to (virtually) have constant delay. Consequently, the delay of the scan path is now independent of π_0.

Figure 6.7 illustrates a modification that operates with lower power. Here all parts of the broadcast network, except the one corresponding to π_0, are disabled. The control signals needed for this network can be generated alongside the generation of h_i as the following. Let d be the *degree* (number of children of each node) of the proposed network with n leaves. This will result in a $\log_d n$-level tree. The process starts at the leaves which hold the h_i values. Each leaf sends its h_i value to its parent. Each internal node x at the first level generates a signal c_x from the OR of the received bits from its d children. This signal controls the tristate present at node x. Each node also sends c_x to its parent and the process

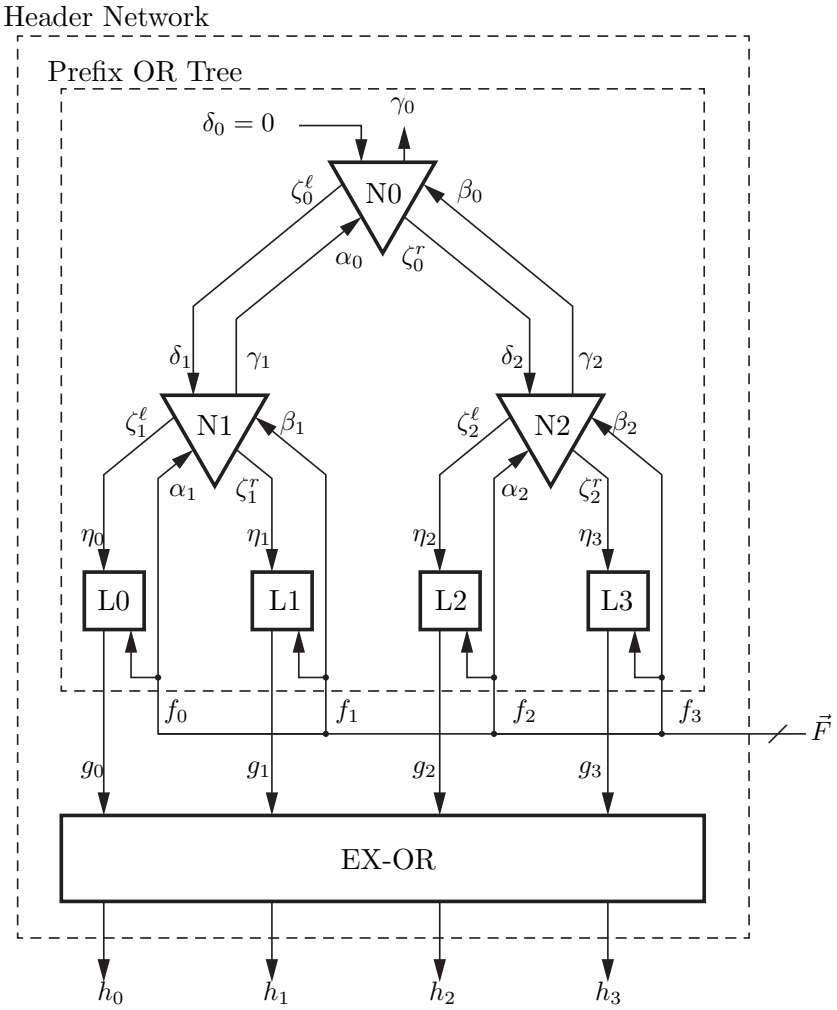


Figure 6.6: Block diagram of the Header Network with $n = 4$. Internal nodes are labeled as Ni and leaf nodes as Li. All the other signals are also labeled accordingly.

repeats for every level recursively until the c_{root} is generated at the root; a process identical to that of Basic and Header Networks for generating the γ_x values. Figure 6.7 shows this network of tristate gates with fan-out 4 (the degree of the tree structure). In practice, such a (non-broadcasting) network can support a fan-out that is larger than that of a clock distribution (broadcast) network and can, as argued earlier, be assumed to be of constant delay. Furthermore, although we elaborated on how to construct a fast header path in this section, later on when we use Verilog HDL in order to implement the design, we will use behavioral code to describe such a network. Therefore, the synthesis tool will decide on how to construct the header path in the most efficient way. In most cases, the computer's solution for such broadcasting networks is similar to that of the clock distribution network

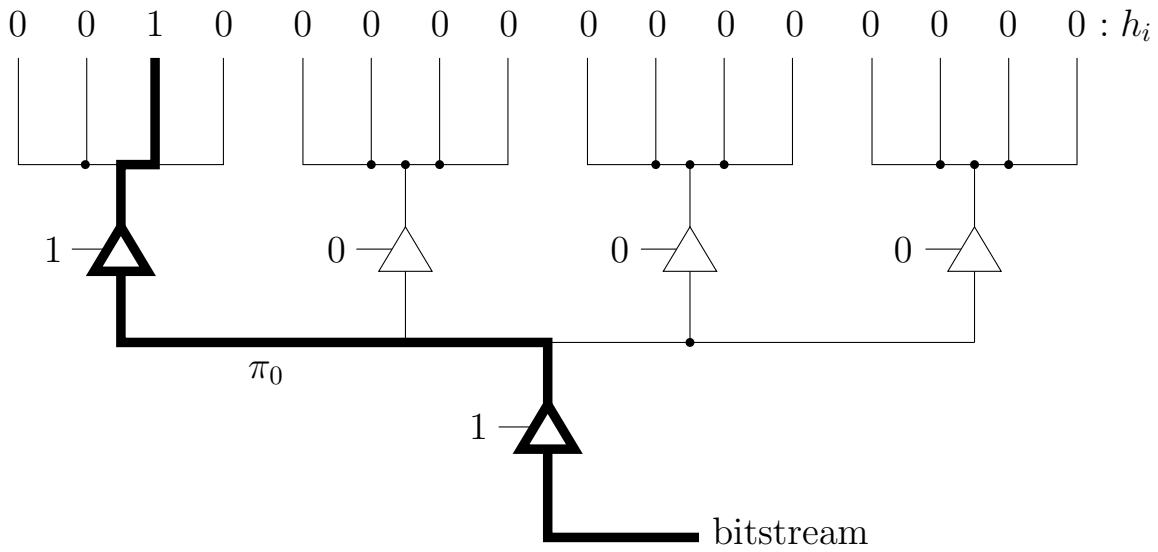


Figure 6.7: An example of establishing the header path π_0 using tristate buffers. The active path is shown in bold.

(also known as the "Clock Tree") spanning the designs to deliver the clock signals to those elements with a clock input. Therefore, the synthesis tool will most probably use a similar network for the header path in our case.

Lemma 11 For any set of k elements selected out of n elements, there exists a Header Network that determines a constant delay path to the first of the selected elements. This network uses $\Theta(n)$ gates and has a delay of $\Theta(\log n)$. ■

Chapter 7
Synthesis and Modeling

In the previous chapters, we developed the proposed ideas theoretically (algorithmically) and detailed possible logic level implementation. An analysis determined that the cost of each of the proposed networks is $O(n)$ with n being the total number of frames or as we will simply call it, the "size" of the network. Additionally, we also argued that for some cases the time to input k configuration bits is $\Theta(k)$. However, knowing only the order of the cost and delay does not indicate the magnitude of the constants hidden by the notation. To better estimate these constants and factor in practical constraints, we utilize the available synthesis tools and implement the proposed ideas to derive equations that model the cost and delay of the networks as a function of n. The networks we synthesize are the following:

The Basic Network: This is the tree illustrated in Figure 3.6 (page 24). The MU-Decoder shown in this figure is only for illustrating the idea of selecting a set of k elements; but to simplify the synthesis that module is replaced by a standard decoder with $\log n$ inputs and n outputs.

The Basic Network with Clock Recommendation: This consists of the Basic Network, the Header Network (Chapter 6), and the Clock Recommendation Network (Section 5.3, page 41).

The Shortcuts Network: This includes the Basic Network augmented with shortcut links (Chapter 4) along with the Header and the Clock Recommendation Network of Section 5.4 (page 45).

The proposed networks are implemented using Verilog HDL and synthesized through the Cadence environment described below. All digital implementations (synthesis) use

tools from Cadence Design Systems suite, namely, (a) Cadence Encounter® RTL Compiler (version RC14.24) for netlist generation and (b) Cadence Encounter Digital Implementation System (EDI) for Place and Route (PAR) [38]. The technology file used is the open access 45nm Process Design Kit (PDK) and cell library "FreePDK45" jointly developed by Oklahoma State University (OSU) and North Carolina State University (NCSU)[39]. Also, MATLAB [40] is used for the interpolation of the data points and mathematical modeling.

We now describe the structure of this chapter. Section 7.1 discusses the implementation (synthesis) of the various networks, whose aim is to determine the delay, area, and power of implementations for different sizes. Section 7.2 includes plots of the derived data from the synthesis phase. Section 7.3 details the modeling phase where the data from Section 7.2 is fitted to obtain an equation modeling the performance of the different networks. Figure 7.1 shows an overarching view of the entire synthesis and modeling phases. To a large extent, the approach in this chapter is similar to that of Raghavendra Kongari's MS thesis [41].

7.1 Synthesis Architecture and Design Flow

The synthesis phase has two stages: (a) netlist generation and (b) place and route (PAR). The top half of Figure 7.1 shows these stages. The netlist generation stage revolves around the RTL Compiler which, given a Verilog code, produces a digital circuit represented as a netlist accompanied by timing, area, and power reports. In particular, the timing report is expressed in terms of a *slack* relative to a clocking constraint. For example, let the clock period of a network be set to 500 ps. Now, assume that the RTL Compiler synthesizes the network with a clock period of 480 ps and a slack of +20 is reported indicating that the clock constraint could be tightened. Similarly, a negative slack indicates an estimate of additional clocking time needed for proper implementation.

The number of synthesis runs for each network is in the order of hundreds (approximately 40 different sizes per each network and several iterations per each size); therefore, it is not possible to perform them manually and without automation. Therefore, we use Tcl

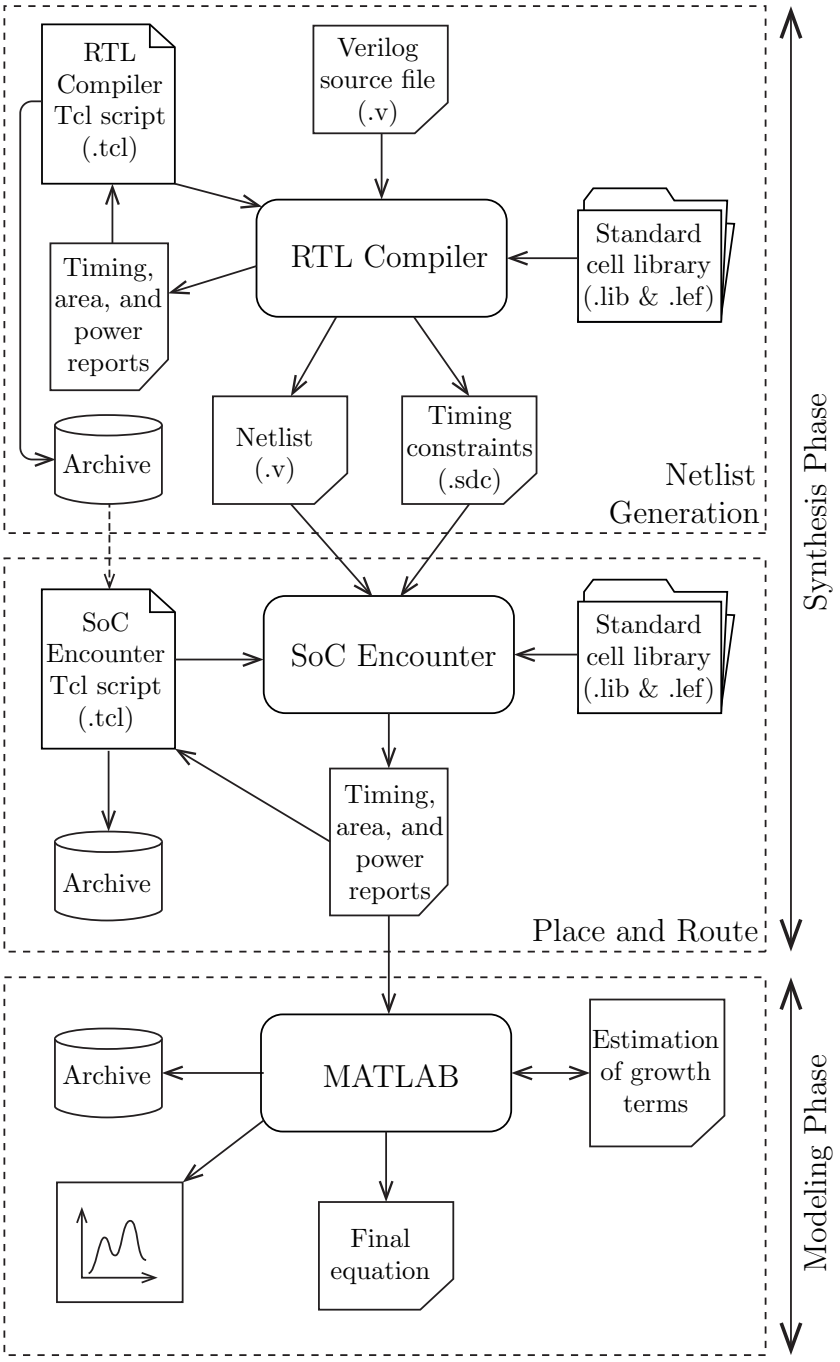


Figure 7.1: An illustration of the design flow.

scripts for each implementation phase. The RTL Compiler Tcl script presents the Verilog code (say for the Basic Network) to the RTL Compiler successively for different network sizes (n from 2^4 to 2^{14}). For each n it examines the timing report to determine if the clock can be tightened. Using a binary search type of algorithm similar to that of Raghavendra's [41] (see Figure 7.2), the Tcl script initiates several synthesis trials for a fixed n until a slack of 0 (or nearly 0 for some cases) is achieved. This ensures the best possible clock period for the network, hence generating a "time-optimized" netlist. At the end of this stage, an output netlist file (with ".v" extension) and a timing constraint file (with ".sdc" extension) are generated and saved for the next stage. Figure 7.3 shows an example logic diagram (i.e. the netlist) generated by RTL Compiler for the Basic Network with $n = 16$ excluding the master controller and the decoder. Table 7.1 shows an example report produced by the RTL Compiler Tcl script on the output terminal while performing synthesis on the Basic Network with $n = 16$. Note that the synthesis tool uses the algorithm shown in Figure 7.2 to achieve zero slack after four iterations.

Table 7.1: Example report generated by the RTL Compiler Tcl script for the Basic Network with $n = 16$.

	Module	Effort	Size (n)	Area (μm^2)	Power (mW)	Clock (ns)	Slack (ns)
1	basic_tree	high	16	1509	15863	50000	49057
2	basic_tree	high	16	1935	1316403	471	-244
3	basic_tree	high	16	1857	910900	707	-32
4	basic_tree	high	16	1498	715040	825	0

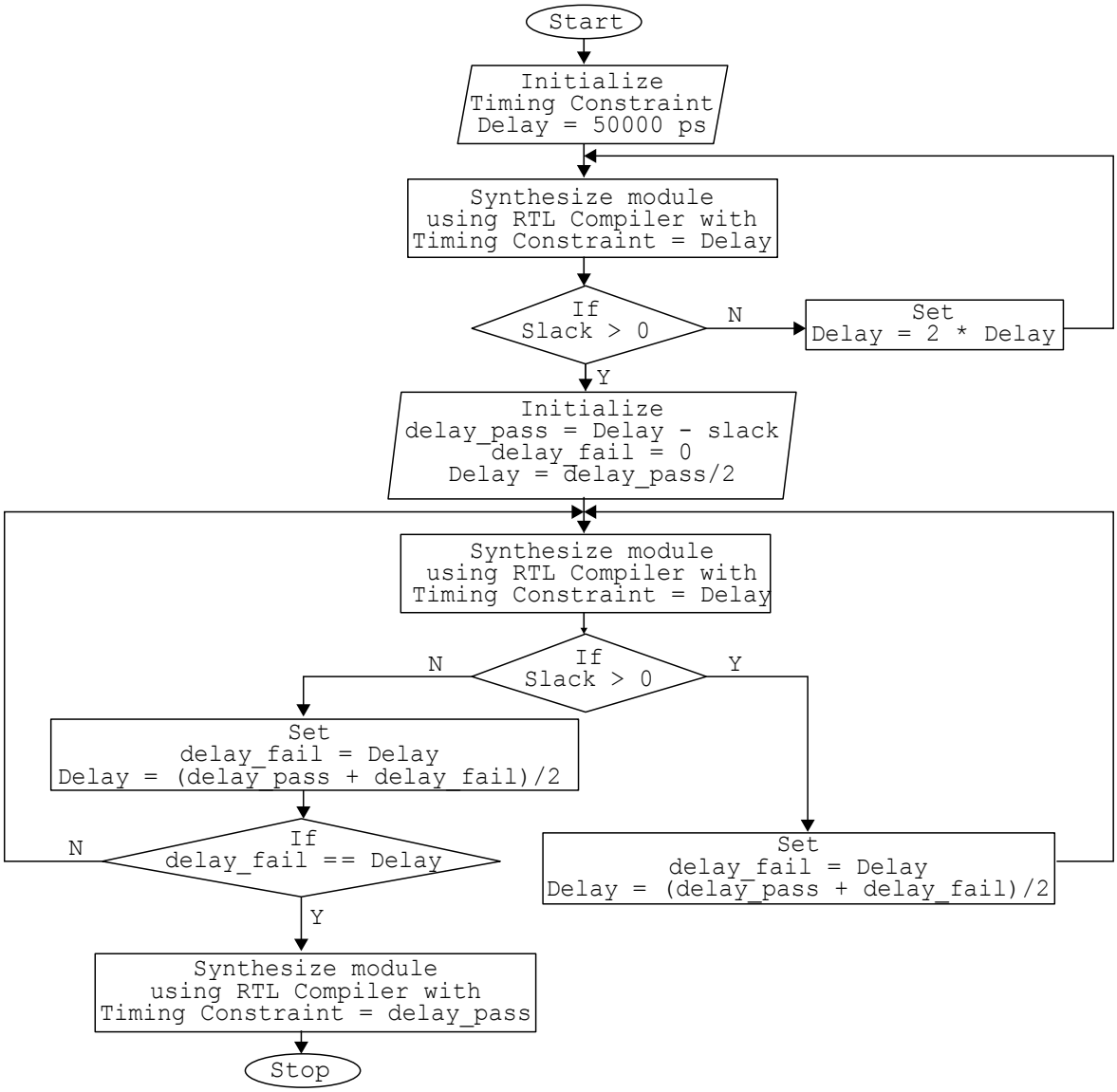


Figure 7.2: Flow chart of the algorithm used in the RTL Compiler Tcl script to optimize the networks with respect to the clock period (i.e., time-optimized). [41].

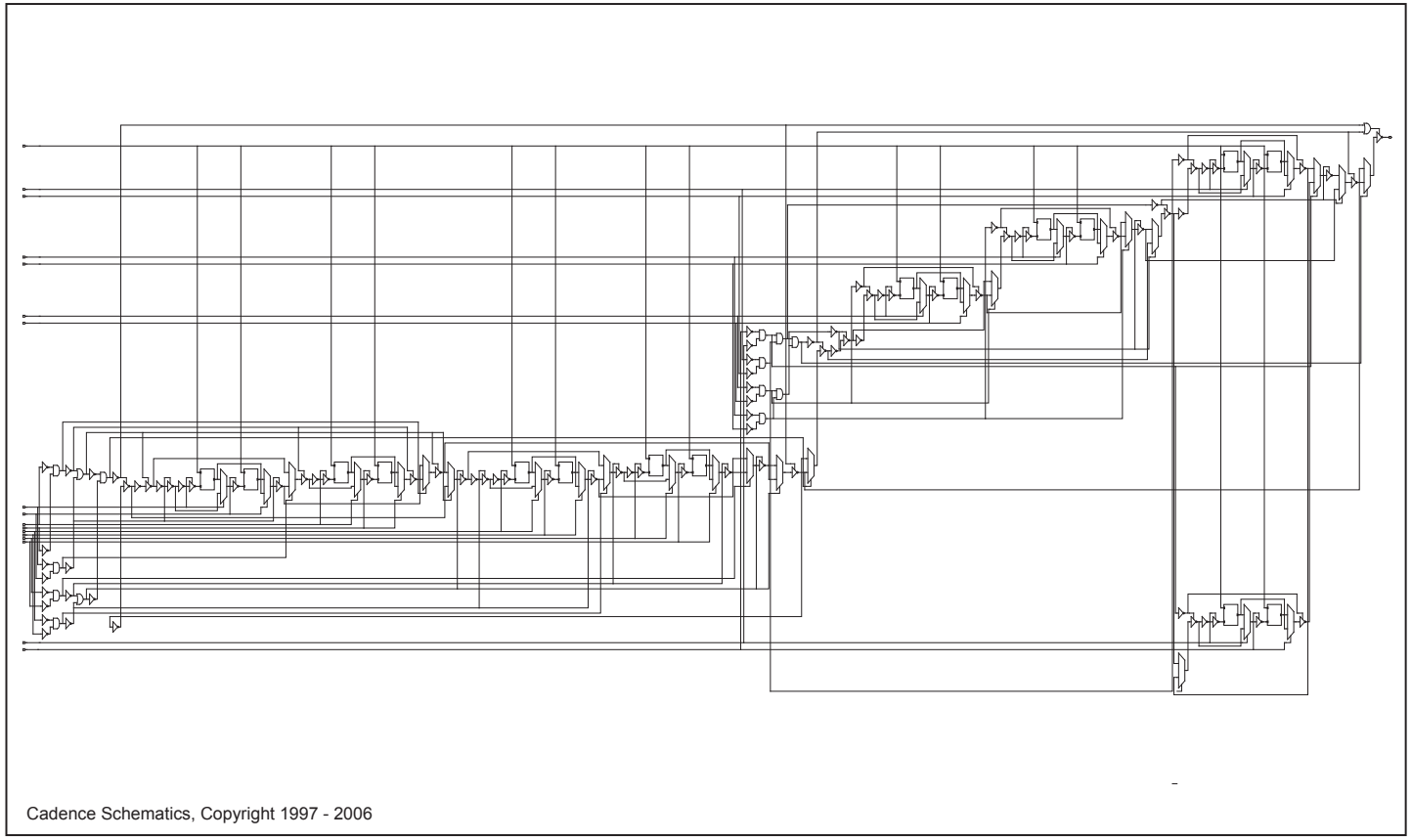


Figure 7.3: Example schematic for the Basic Network with $n = 16$. The master controller and the decoder are not included.

At the end of the netlist generation stage, the netlist and timing constraint files are generated for each size and stored so that the SoC Encounter Tcl script can access them for the place and route stage. In the PAR stage, all the physical aspects of the circuit such as wire lengths, parasitic elements (e.g. capacitances), voltage drops, power leaks, etc., are considered while they are not taken into account in the previous stage. As in the netlist generation, the PAR stage is also automated with a Tcl script that uses iterations to make sure the final output meets the timing constraints. In particular, the SoC Encounter Tcl script iterates through different network sizes and performs the following tasks automatically (a) adjusts the paths of the input files and sets other applicable variables (e.g. the distance between the power grids), (b) performs the steps of physical design (such as floor planning, power planning, placing, routing, clock tree generation, timing analysis and optimization), (c) generates the reports and archives them properly, (d) extracts the desired timing, area, and power values from report files, and ultimately (e) generates a report file and accumulates all the extracted results for all sizes. Also for each size, the script checks the value of the"worst negative slack" in the reports and initiates iterations to resolve the negative slack by relaxing the clock period. Figure 7.4 shows an example layout of the Shortcuts Network with $n = 16$, generated at the end of the PAR stage.

7.2 Synthesis Results

As detailed in Section 7.1, each network (Basic, Basic with Clock Recommendation, and Shortcuts Network) was implemented for a range of values of n (network size)and optimized for time using Tcl scripts. For each implementation we collected data on clock period (T), area (A), and power (P) and plotted the each one with respect to size (n). n is set by the script such that $n \in \{\lfloor 2^{i/4} \rfloor : 16 \leq i \leq 56\}$. This will result in the sizes to be powers of 2 (from 2^4 to 2^{14}) plus three more data points distributed exponentially in between each two powers of 2 (e.g. $\lfloor 2^{8.25} \rfloor = 304$, $\lfloor 2^{8.5} \rfloor = 362$, and $\lfloor 2^{8.75} \rfloor = 430$ between 256 and 512). T plotted here is the best achievable clock period for each network.

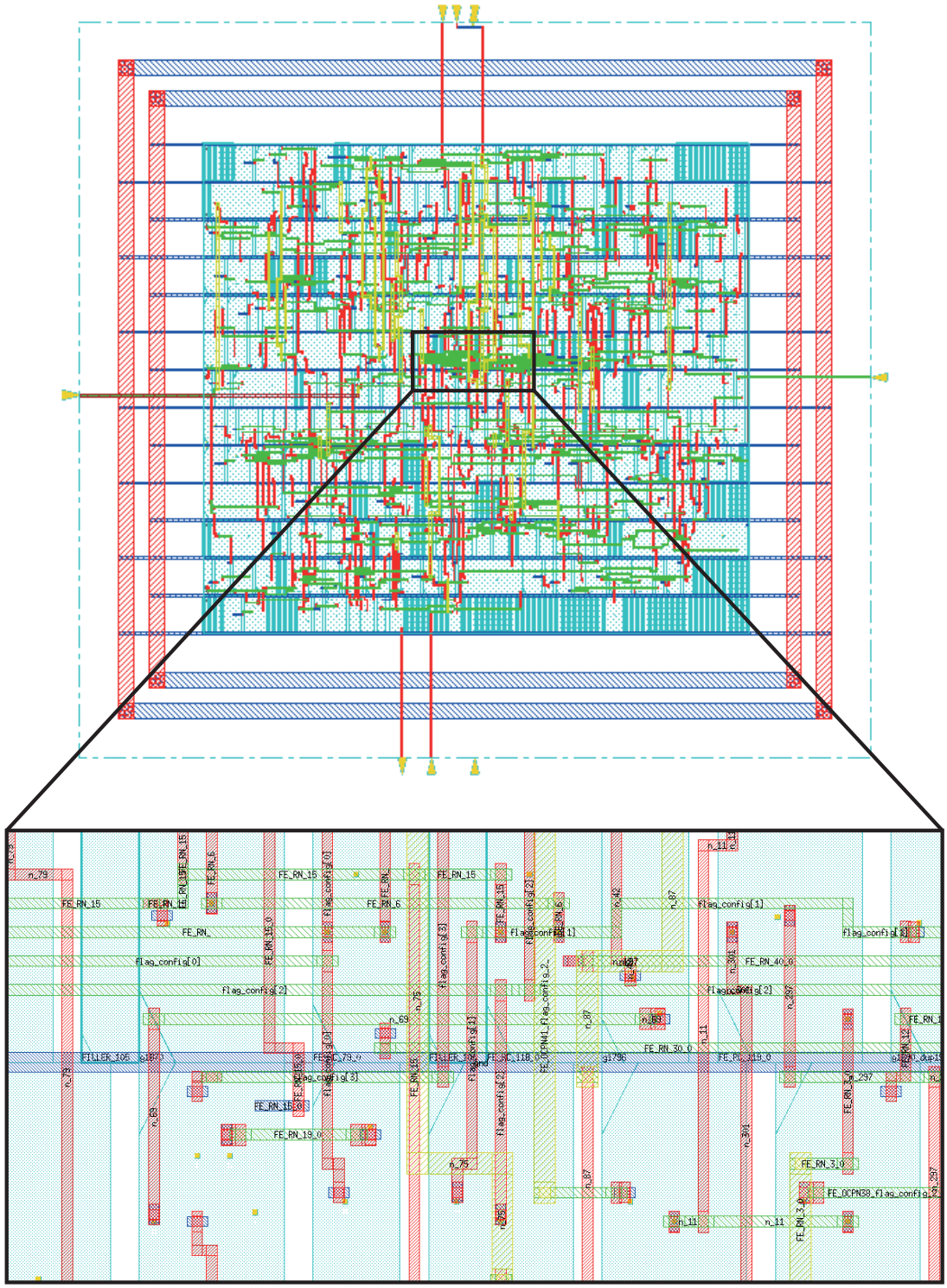


Figure 7.4: Example layout for the Shortcuts Network with $n = 16$ generated in SoC Encounter.

Considering the the worst case delay of $O(\log n)$ in our theoretical analysis, we expect all our designs to have logarithmic delay with respect to size. Therefore, each of the timing graphs are also expressed in logarithmic scale. Figures 7.5 to 7.16 show the results for the proposed networks. Dashed lines show the curves fitted to the data from Section 7.3.

Although we will quantitatively examine the growth terms of these graphs in Section 7.3, it is easy to see that the differences in cost and performance between the Basic Network and the Basic Network with clock recommendation is reasonably small (see Figures 7.17 to 7.19). This and other performance comparisons between networks are made in the Section 7.4.

7.3 Modeling Stage

At the end of the synthesis stage, we have a set of reports containing data for clock period, area, and power for each network and each n generated by the SoC Encounter Tcl script. For example for the Basic Network with clock recommendation, Table 7.2 shows a part of the data also captured in the plots of Figures 7.9 to 7.12. The idea is to use this data to derive functions $\mathcal{T}_B(n)$, $\mathcal{A}_B(n)$, and $\mathcal{P}_B(n)$ that indicate the clock period, area, and power of the Basic Network as a function of its size n. We use the Curve Fitting Toolbox $^{\text{TM}}$(CFT) in MATLAB to curve fit this data. For the Basic Network, the theoretical estimate for the clock period is $O(\log n)$. However, a layout with n leaves and $\log n$ levels could include wires of length $O(n)$ and $O(\log n)$ which could lead to a delay proportional to these terms. Using a similar rationale, area and power may be proportional to n^2, $n \log n$, or $\log^2 n$. Thus, we included 1, $\log n$, n, $\log^2 n$, $n \log n$, and n^2 as target growth terms. After the first attempt in the CFT, some of these growth terms had nearly 0 coefficients so we repeat the curve fitting without them. Specifically for $\mathcal{T}_B(n)$, the initial general equation was

$$\mathcal{T}_B(n) = a \cdot n^2 + b \cdot n \log n + d \cdot n + c \cdot \log^2 n + e \cdot \log n + f$$

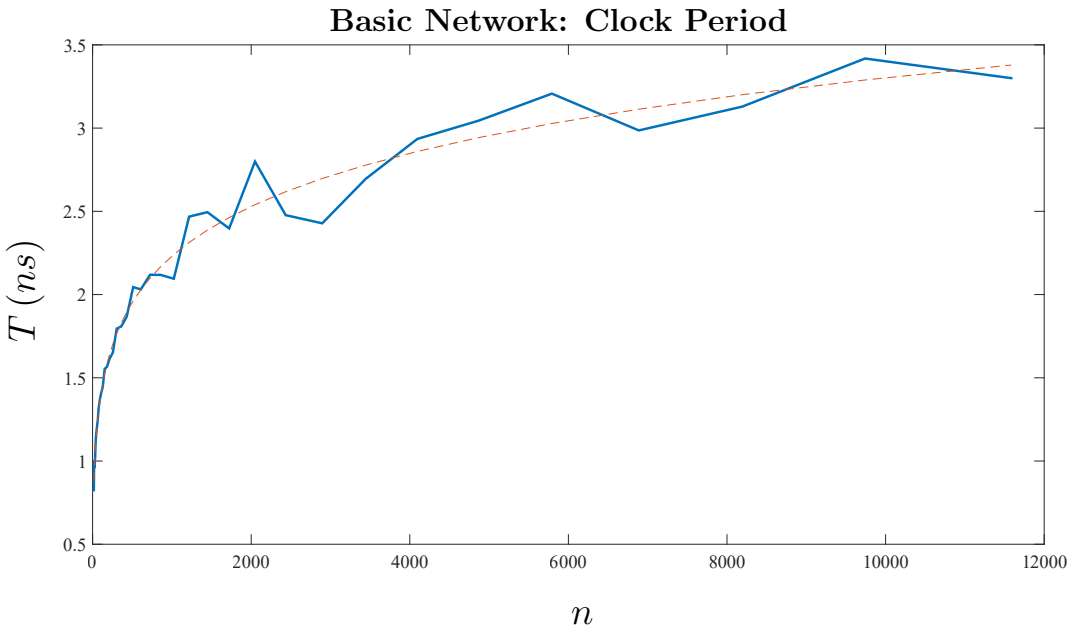


Figure 7.5: Clock period (T) results for the Basic Network as a function of size n. Dashed line shows the result of the curve fitting from Section7.3 (similar for Figures7.6 to 7.16).

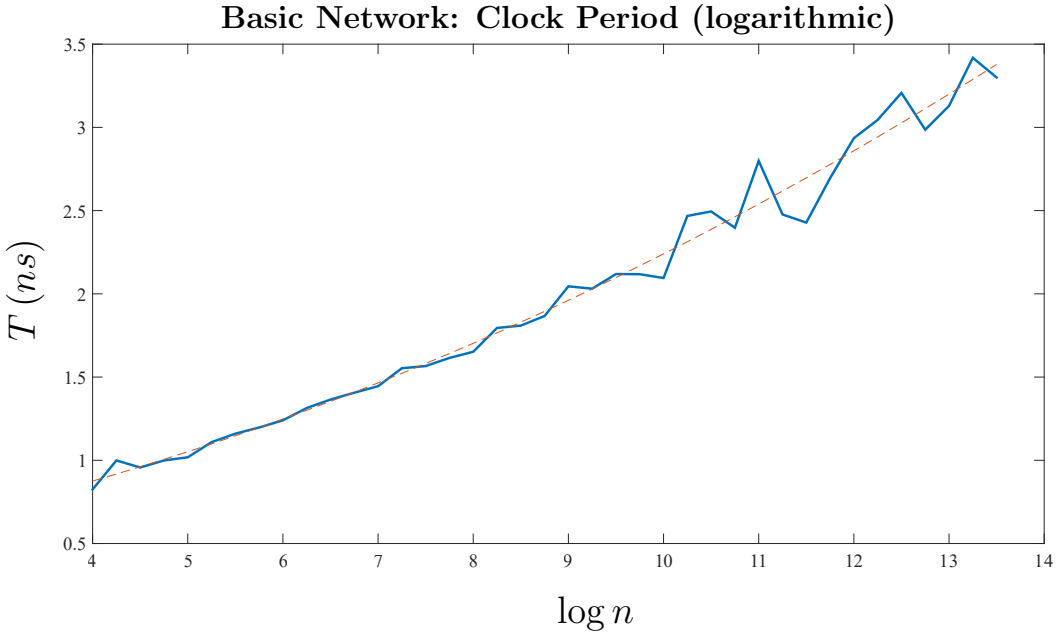


Figure 7.6: Clock period (T) results for the Basic Network as a function of $\log n$ (logarithmic x-axis).

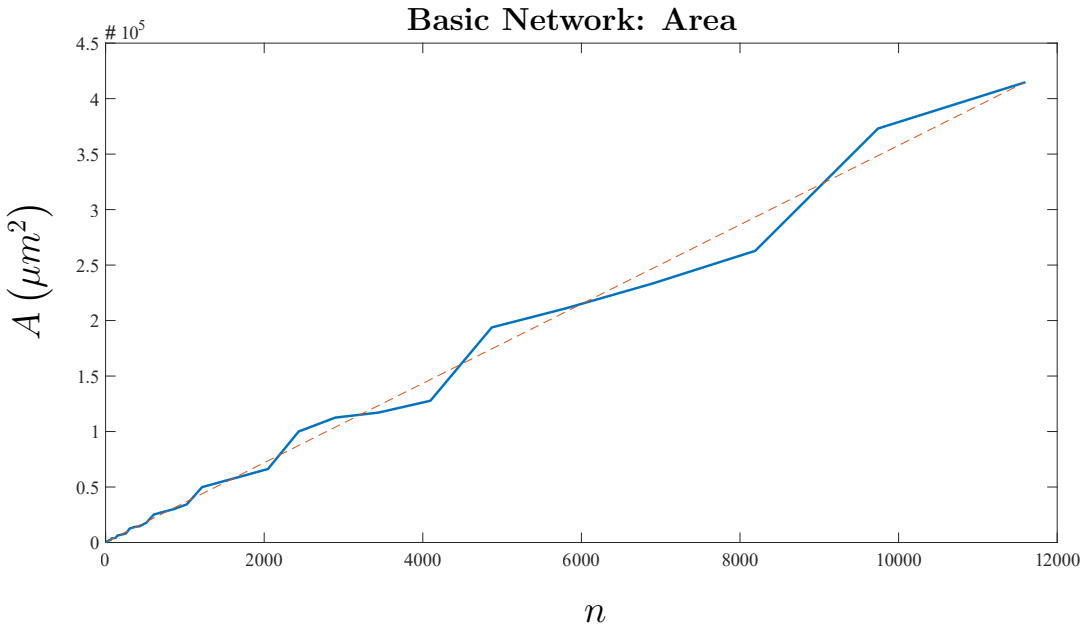


Figure 7.7: Area (A) results for the Basic Network as a function of size n.

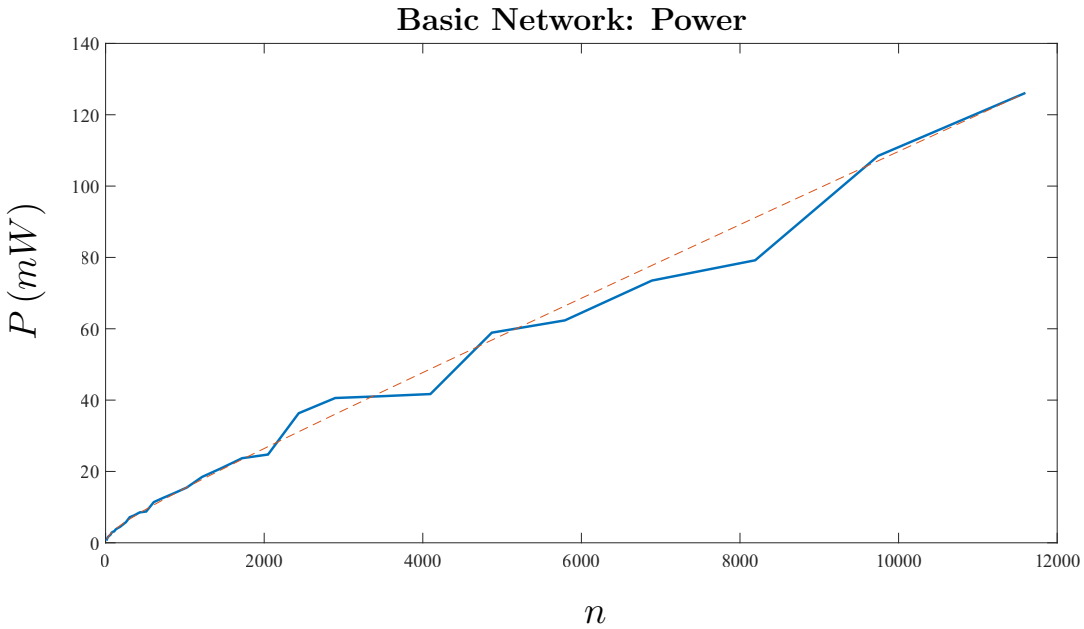


Figure 7.8: Power (P) results for the Basic Network as a function of size n.

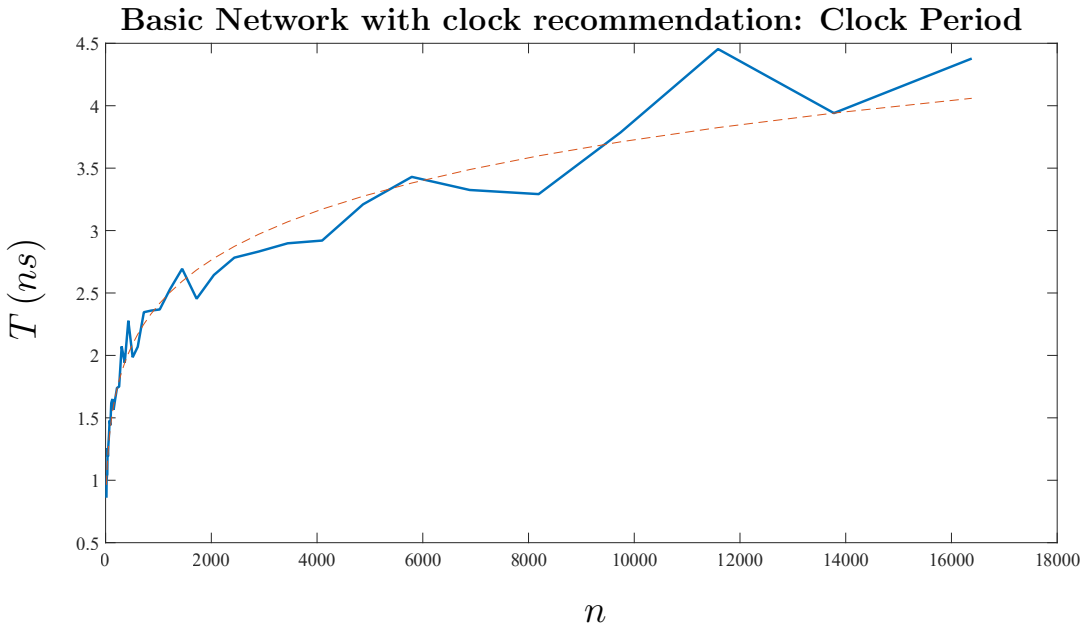


Figure 7.9: Clock period (T) results for the Basic Network with clock recommendation as a function of size n.

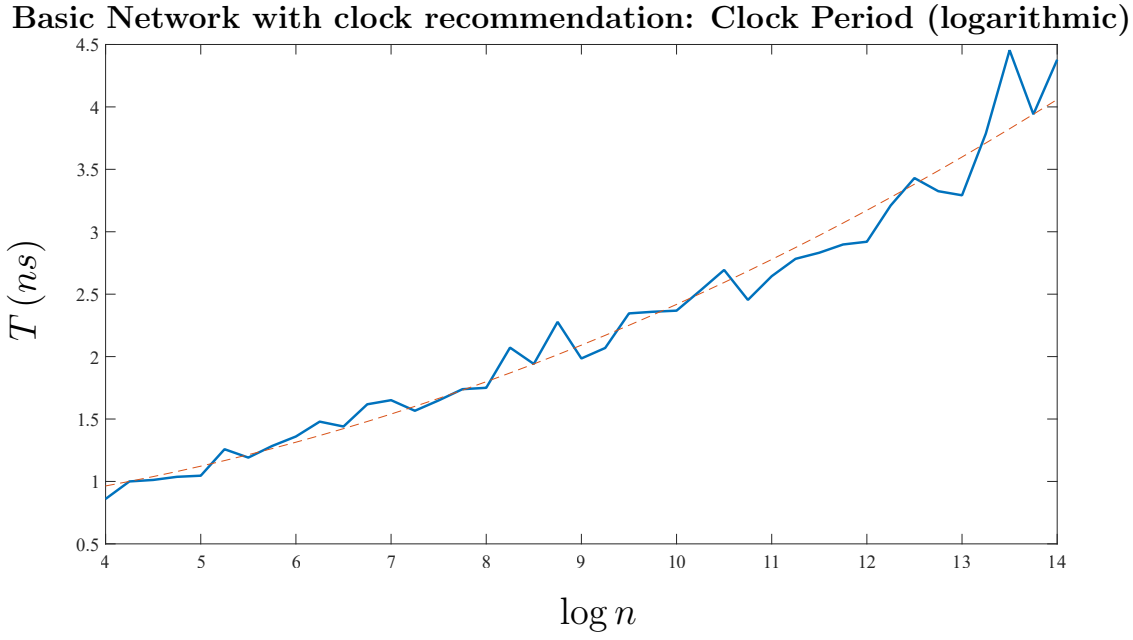


Figure 7.10: Plot for Basic Network with clock recommendation, clock period T vs. logarithmic size $\log n$.

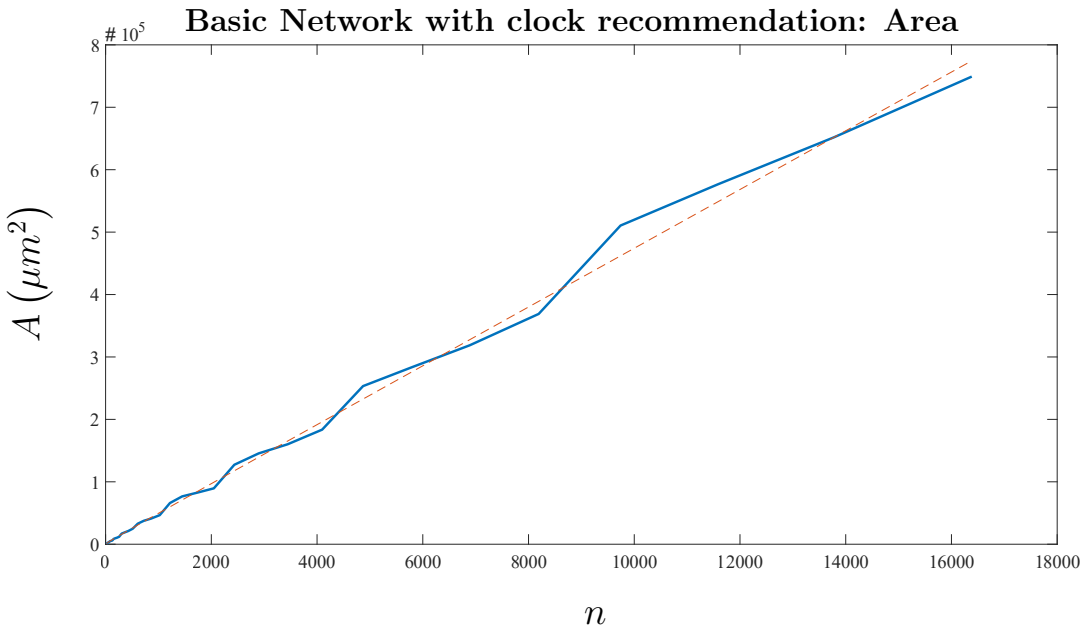


Figure 7.11: Area (A) results for the Basic Network with clock recommendation as a function of size n.

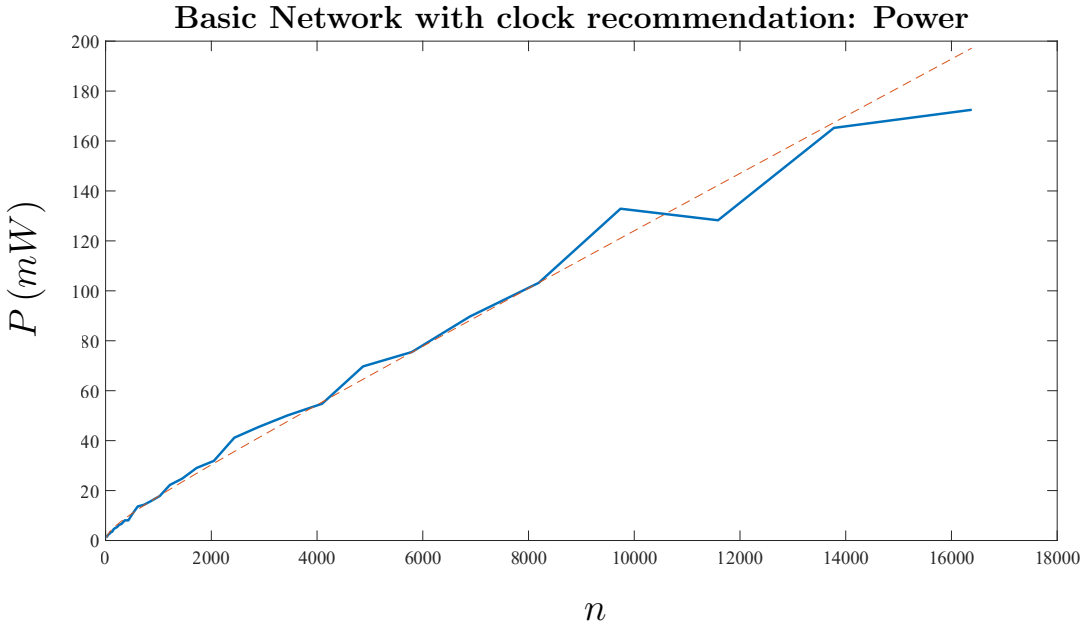


Figure 7.12: Power (P) results for the Basic Network with clock recommendation as a function of size n.

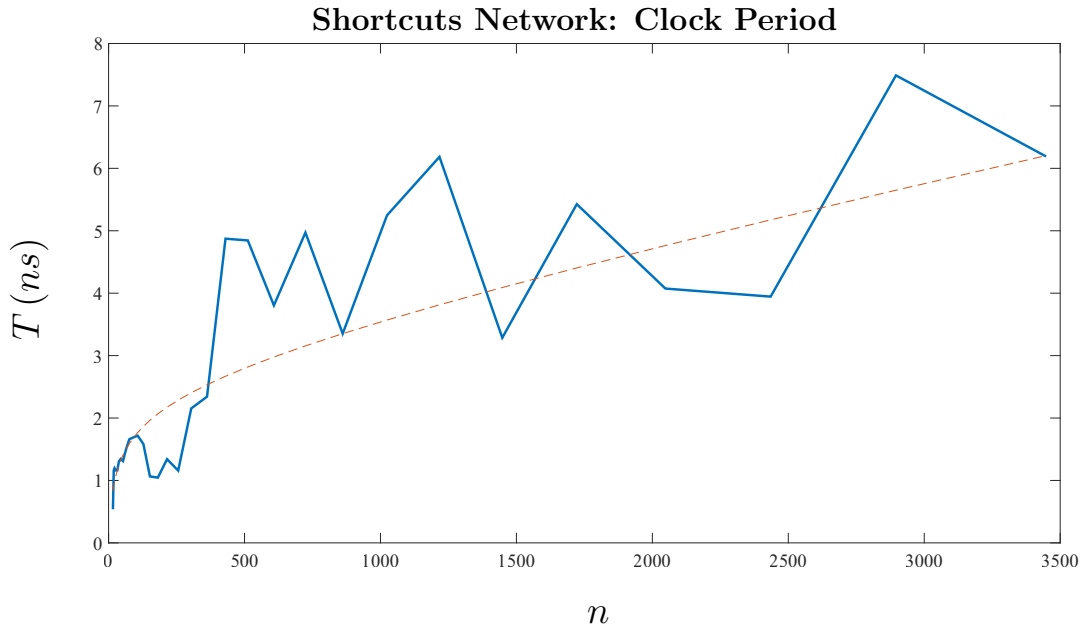


Figure 7.13: Clock period (T) results for the Shortcuts Network as a function of size n.

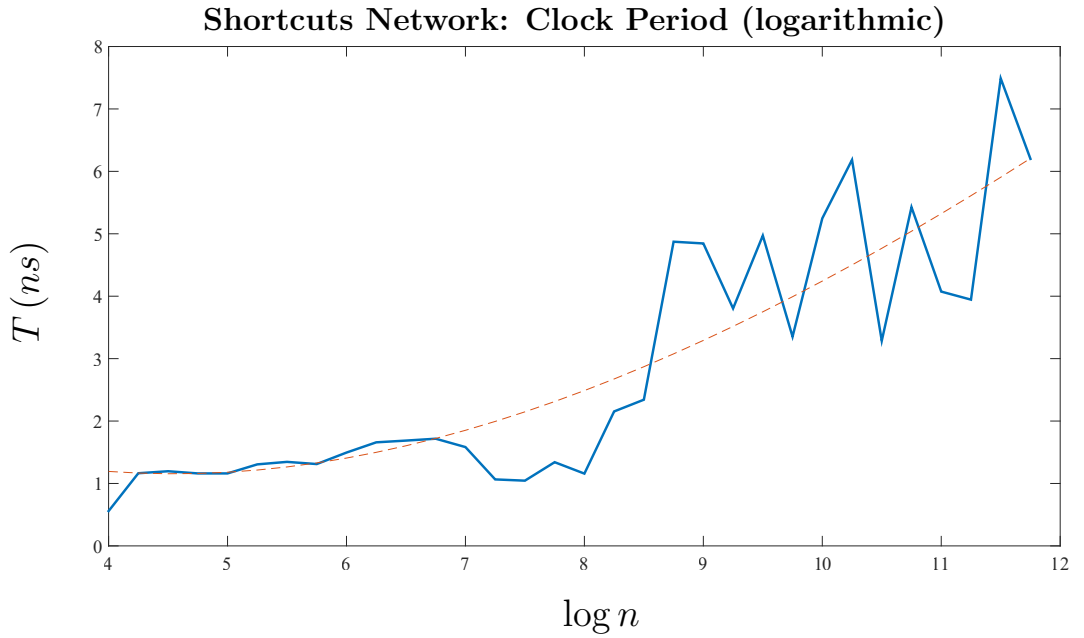


Figure 7.14: Clock period (T) results for the Shortcuts Network as a function of $\log n$ (logarithmic x-axis).

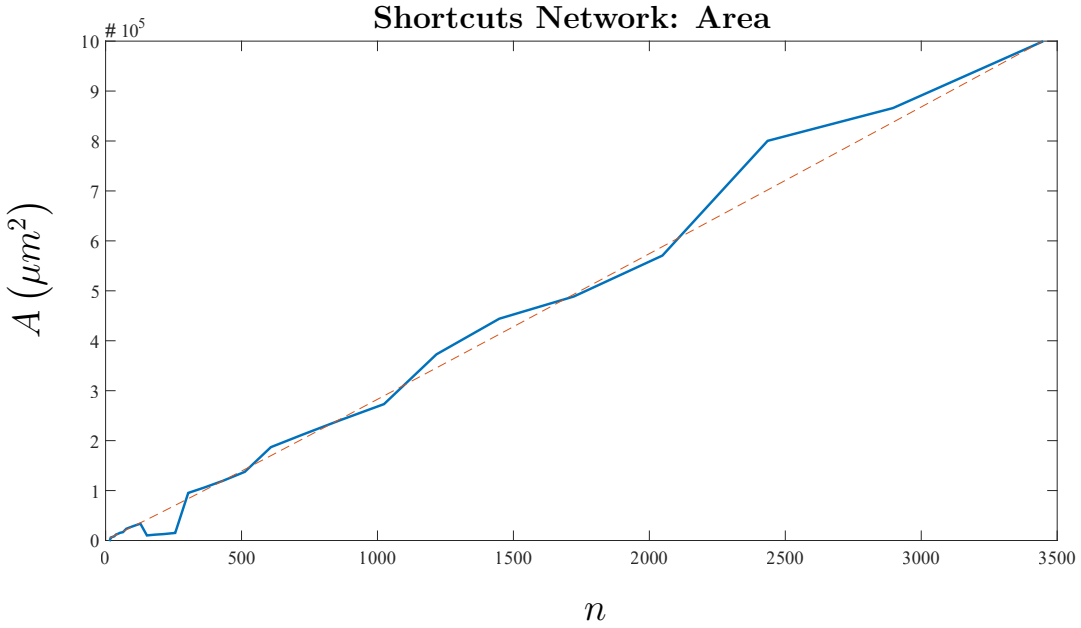


Figure 7.15: Area (A) results for the Shortcuts Network as a function of size n.

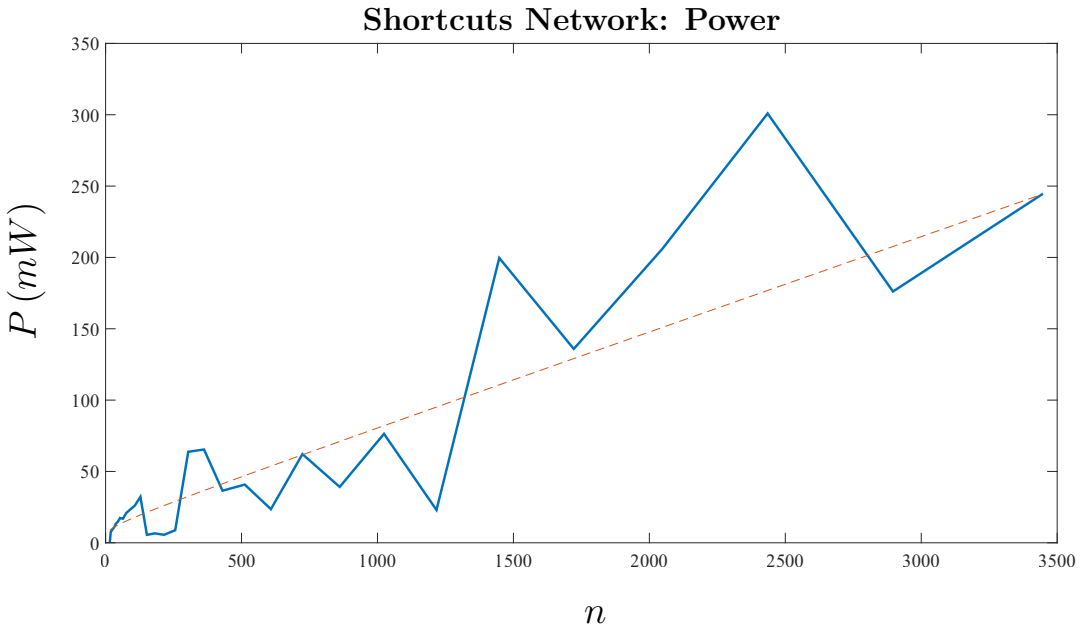


Figure 7.16: Power (P) results for the Shortcuts Network as a function of size n.

Table 7.2: A part of the final report generated at the end of the synthesis stage for the Basic Network with clock recommendation.

Size (n)	Clock Period	Area (μm^2)	Power (mW)	Slack (ns)
⋮	⋮	⋮	⋮	⋮
724	2.346	37566.0571	14.29416028	0.091
861	2.359	41111.1493	15.82467362	0.089
1024	2.368	46603.8365	17.81097354	0.002
1217	2.529	65977.0098	22.27043528	0.020
1448	2.694	76604.3083	24.78037995	0.022
1722	2.454	82379.5141	29.05829671	0.023
2048	2.644	89545.2558	31.89152006	0.006
2435	2.783	127298.5636	41.17718596	0.003
2896	2.832	145527.5835	45.48067184	0.067
3444	2.898	160145.3399	50.11557636	0.011
4096	2.920	183548.8616	54.78606798	0.006
⋮	⋮	⋮	⋮	⋮

which after curve fitting resulted in

$$\mathcal{T}_B(n) = \left(1.859 \times 10^{-12}\right) n^2 + \left(1.061 \times 10^{-9}\right) n \log n + (0.009228) \log^2 n + \\ + \left(3.552 \times 10^{-14}\right) n + (0.09339) \log n + 0.357.$$

Its clear that the n^2, $n \log n$, and n terms have nearly 0 coefficients and $\mathcal{T}_B(n) = c \cdot \log^2 n + e \cdot \log n + f$ would be a better fit. Therefore, the updated equation was again run through the CFT to obtain:

$$\boxed{\mathcal{T}_B(n) = 0.01035 \log^2 n + 0.08205 \log n + 0.3848.} \tag{7.1}$$

Similarly after adjustments of coefficients, the curve fitting for $\mathcal{A}_B(n)$ and $\mathcal{P}_B(n)$ yields the following equations:

$$\boxed{\mathcal{A}_B(n) = 35.73n + 2.004 \log^2 n + 176.1} \tag{7.2}$$

$$\mathcal{P}_B(n) = 0.01004n + 0.05283\log^2 n \tag{7.3}$$

We repeat this process for the Basic Network with clock recommendation (BC) and the Shortcuts Network (S) to obtain $\mathcal{T}_{BC}(n)$, $\mathcal{A}_{BC}(n)$, $\mathcal{P}_{BC}(n)$, $\mathcal{T}_S(n)$, $\mathcal{A}_S(n)$, and $\mathcal{P}_S(n)$ as follows:

$$\mathcal{T}_{BC}(n) = 0.0168\log^2 n + 0.007154\log n + 0.6665 \tag{7.4}$$

$$\mathcal{A}_{BC}(n) = 46.96n + 25.9\log^2 n \tag{7.5}$$

$$\mathcal{P}_{BC}(n) = 0.01126n + 0.06487\log^2 n \tag{7.6}$$

$$\mathcal{T}_S(n) = 0.0008621n + 0.3106\log n - 0.419 \tag{7.7}$$

$$\mathcal{A}_S(n) = 298.2n - 585.9\log^2 n + 5656\log n - 1.363 \times 10^{+4} \tag{7.8}$$

$$\mathcal{P}_S(n) = 0.06621n + 1.08\log n + 3.473 \tag{7.9}$$

7.4 Comparison of Network Costs

From Equations 7.1 to 7.9, generally, we have the following results:

(a) the implemented networks have a polylogarithmic delay (clock period) as it was expected from the theoretical analysis. The only exception is the n term (with a relatively small coefficient) that has appeared in Equation 7.7 for the Shortcuts Network's

clock period. The reason behind having such term is that the synthesis tool is not aware of the network's function. Specifically, the Shortcuts Network is designed so that the length of a shortcut path cannot exceed $\log n$. However, there exists a virtual combinational shortcut path of length n that connects all the leaves. Therefore when performing timing analysis, the synthesis tool will take into account those paths with $O(n)$ length.

(b) The area and power of the synthesized networks are $O(n)$ (linear) as expected.

Note that the constants of the derived equations will change when synthesizing with a different technology. Therefore, these equations only provide a sense of relative performance and cost. For example from Equations 7.1 and 7.4, it can be seen that adding the clock recommendation to the Basic Network has a relatively small effect on its delay. Figures 7.17, 7.18, and 7.19 show comparative the plots for clock period (T), area (A), and power (P) of the synthesized networks, respectively.

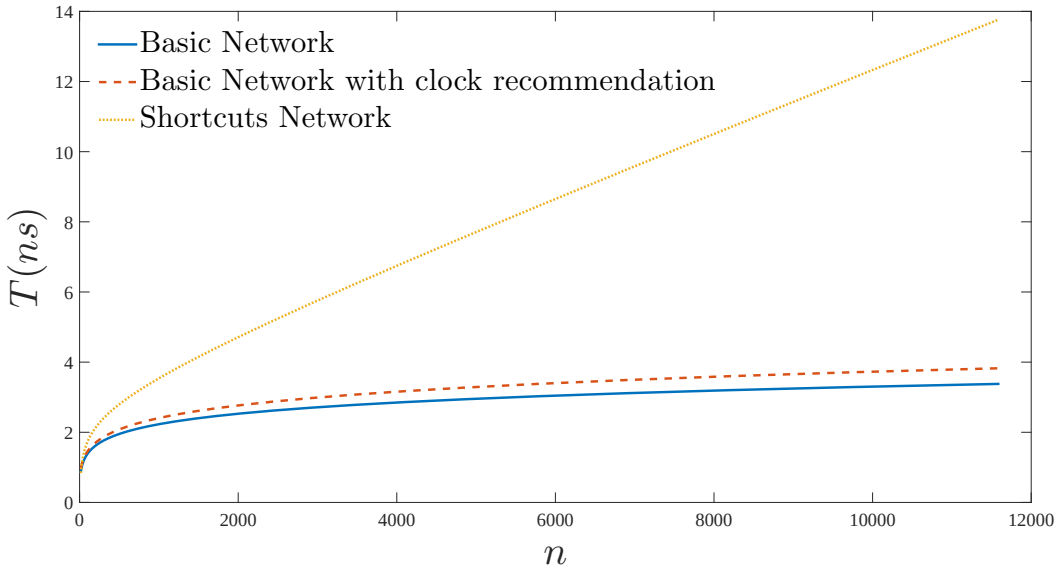


Figure 7.17: Clock period T as a function of n for the synthesized networks.

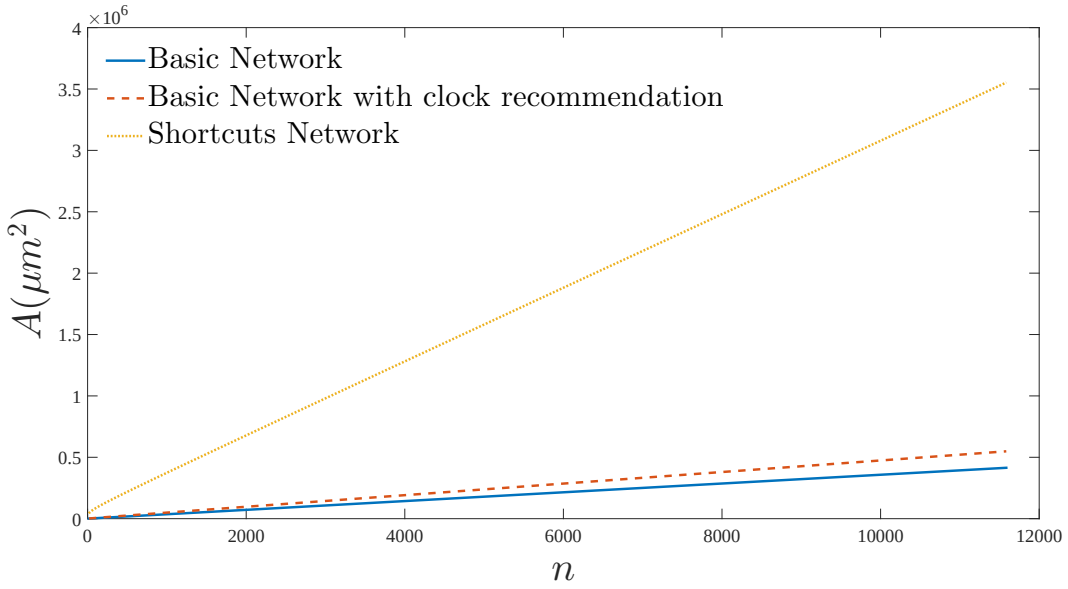


Figure 7.18: Area A as a function of n for the synthesized networks.

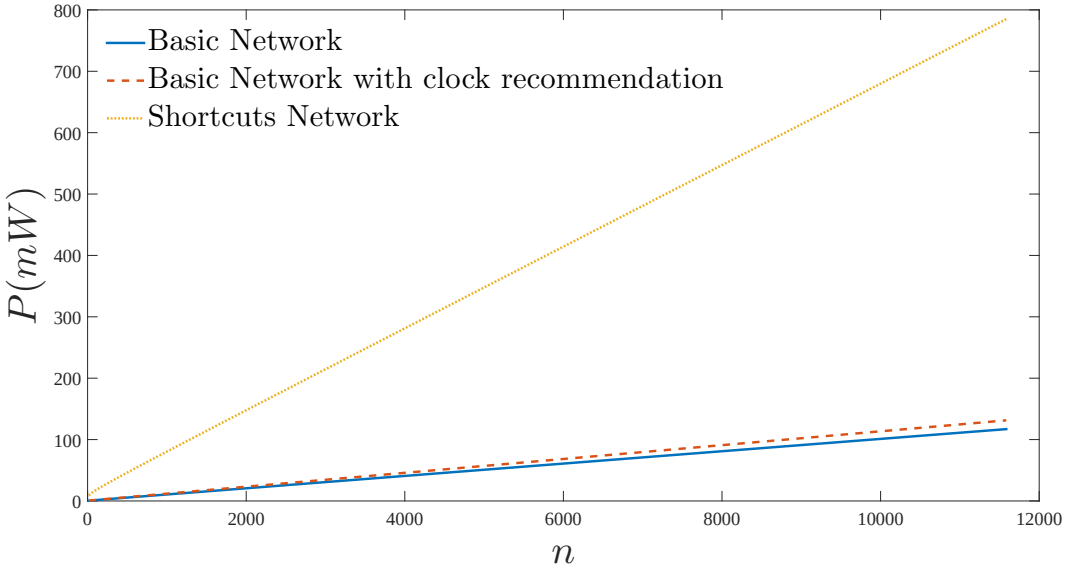


Figure 7.19: Power P as a function of n for the synthesized networks.

Chapter 8
Concluding Remarks

In this thesis, we addressed the problem of making a path to efficiently scan a configuration bitstream into a set of k selected frames out of n total frames. As solutions to this problem, we proposed a class of $\Theta(n)$ cost networks as the following: (a) the Basic Network which uses an underlying binary tree to set a physical scan path connecting the selected frames and scans in the bitstream in $\Theta(k \log n)$, (b) the Shortcuts Network which uses an additional path along the frames to shortcut the tree paths whenever appropriate and can scan the k-bit configuration bitstream in $\Theta(\log n \log\log\log n + kT_0)$ time, where T_0 can be as small as a constant and at most $O(\log n)$, (c) the Clock Recommendation Network, an augmentation to the previous networks which gives an estimate of the best possible admissible clock period, and (d) the Header Network, another enhancement to the proposed networks which channels a direct path to the first selected frame and eliminates the bottleneck of $O(\log n)$ delay. Without the Header Network, it is not possible to exploit the benefits of the Shortcuts and Clock Recommendation networks.

We theoretically showed that the proposed networks work very well by having an optimal cost of $\Theta(n)$; and in some cases, the optimal delay of $\Theta(k)$. We also synthesized the proposed networks and confirmed having polylogarithmic delays and linear costs (area and power) by deriving a mathematical model of delay and cost as a function of network size. It is important to note that the control and data planes are not distinguished in the synthesis. While the control plane includes detailed circuitry, for example, to choose the shorter path between tree or shortcuts in the Shortcuts Network, the additional delay it introduces should not be taken into account when measuring the reconfiguration speed. Thus, the delays are expected to be even less for the data plane and it might be beneficial to perform synthesis for the data plane and the control plane separately.

The work opens up several directions for future research. For example, we can study the same ideas on a non-binary tree (e.g. a ternary or even an m-ary tree). Also, investigating a combination of random and contiguous distribution of k selected frames over n can be helpful. Moreover, using latches instead of flip-flops in the path-length assessment phase of the Shortcuts Network may be beneficial to give a better sense of the (combinational) path delays due to tree and shortcut edges. Additionally, we can try optimizing the networks with respect to area (or power) to include the cases where manufacturing cost is the primary priority.

References

[1] H. S. Neoh and A. Hazanchuk, "Adaptive Edge Detection for Real-time Video Processing Using FPGAs," *Global Signal Processing*, vol. 7, no. 3, pp. 2–3, 2004.

[2] P. Greisen, S. Heinzle, M. Gross, and A. P. Burg, "An FPGA-based Processing Pipeline for High-definition Stereo Video," *EURASIP Journal on Image and Video Processing*, vol. 2011, no. 1, pp. 1–13, 2011.

[3] N. Srivastava, J. L. Trahan, R. Vaidyanathan, and S. Rai, "Adaptive Image Filtering Using Run-time Reconfiguration," in *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*. IEEE, 2003, pp. 7–pp.

[4] S. Bishop, S. Rai, B. Gunturk, J. L. Trahan, and R. Vaidyanathan, "Reconfigurable Implementation of Wavelet Integer Lifting Transforms for Image Compression," in *Reconfigurable Computing and FPGA's, 2006. ReConFig 2006. IEEE International Conference on*. IEEE, 2006, pp. 1–9.

[5] H. Song and J. W. Lockwood, "Efficient Packet Classification for Network Intrusion Detection Using FPGA," in *Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-programmable Gate Arrays*, ser. FPGA '05. New York, NY, USA: ACM, 2005, pp. 238–245. [Online]. Available: http://doi.acm.org/10.1145/1046192.1046223

[6] C. Sao-Jie, Y.-C. Lan, R. Wen-Chung, and H. Yu-Hen, *Reconfigurable Networks-on-Chip*, 1st ed. Springer Publishing Company, Incorporated, 2012.

[7] M. S. Abdelfattah, A. Bitar, and V. Betz, "Take the Highway: Design for Embedded NoCs on FPGAs," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '15. New York, NY, USA: ACM, 2015, pp. 98–107. [Online]. Available: http://doi.acm.org/10.1145/2684746.2689074

[8] M. K. Papamichael and J. C. Hoe, "CONNECT: Re-examining Conventional Wisdom for Designing NoCs in the Context of FPGAs," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '12. New York, NY, USA: ACM, 2012, pp. 37–46. [Online]. Available: http://doi.acm.org/10.1145/2145694.2145703

[9] J. W. Lockwood, A. Gupte, N. Mehta, M. Blott, T. English, and K. Vissers, "A Low-latency Library in FPGA Hardware for High-frequency Trading (HFT)," in *High-Performance Interconnects (HOTI), 2012 IEEE 20th Annual Symposium on*. IEEE, 2012, pp. 9–16.

[10] S. Parsons, D. Taylor, D. Schuehler, M. Franklin, and R. Chamberlain, "High Speed Processing of Financial Information Using FPGA devices," Mar. 26 2013, uS Patent 8,407,122. [Online]. Available: https://www.google.com/patents/US8407122

[11] M. B. Gokhale and P. S. Graham, *Reconfigurable Computing: Accelerating Computation with Field-Programmable Gate Arrays*, 1st ed. Springer Publishing Company, Incorporated, 2010.

[12] C. Bobda, *Introduction to Reconfigurable Computing: Architectures, Algorithms, and Applications*, 1st ed. Springer Publishing Company, Incorporated, 2007.

[13] I. Kuon, R. Tessier, and J. Rose, "FPGA Architecture: Survey and Challenges," *Found. Trends Electron. Des. Autom.*, vol. 2, no. 2, pp. 135–253, Feb. 2008.

[14] H. Chen, Y. Chen, and D. H. Summerville, "A Survey on the Application of FPGAs for Network Infrastructure Security," *Communications Surveys & Tutorials, IEEE*, vol. 13, no. 4, pp. 541–561, 2011.

[15] D. Kock, *Partial Reconfiguration on FPGAs: Architectures, Tools and Applications*, 1st ed. Springer Publishing Company, Incorporated, 2013.

[16] K. Vipin and S. A. Fahmy, "ZyCAP: Efficient Partial Reconfiguration Management on the Xilinx Zynq," *IEEE Embedded Systems Letters*, vol. 6, no. 3, pp. 41–44, September 2014.

[17] K. Papadimitriou, A. Dollas, and S. Hauck, "Performance of Partial Reconfiguration in FPGA Systems: A Survey and a Cost Model," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 4, no. 4, pp. 1–24, Dec. 2011.

[18] Xilinx, *Partial Reconfiguration User Guide*, 2013.

[19] *Virtex Series Configuration Architecture User Guide*, Xilinx, 10 2004, table 3.

[20] R. Vaidyanathan, "The MU Decoder and Scan-Path Generation: A Different Approach to FPGA Reconfiguration," 5 2015, presented at 29th IEEE International Parallel and Distributed Processing Symposium, Hyderabad, India.

[21] M. C. Jordan and R. Vaidyanathan, "MU-decoders: A Class of Fast and Efficient Configurable Decoders," in *24th IEEE International Symposium on Parallel and Distributed Processing, IPDPS*, 2010, pp. 1–4. [Online]. Available: http://dx.doi.org/10.1109/IPDPSW.2010.5470731

[22] R. Vaidyanathan and M. C. Jordan, "Configurable Decoder with Applications in FPGAs," Oct. 2014, uS Patent 8862854.

[23] S. D. Brown and Z. G. Vranesic, *Fundamentals of Digital Logic with Verilog Design*, 3rd ed. New York, NY, USA: McGraw-Hill Education, Inc., 2013.

[24] M. Imhof and H.-J. Wunderlich, "Bit-Flipping Scan–A Unified Architecture for Fault Tolerance and Offline Test," in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, March 2014, pp. 1–6.

[25] Y. Bonhomme, P. Girard, L. Guiller, C. Landrault, and S. Pravossoudovitch, "A Gated Clock Scheme for Low Power Scan Testing of Logic ICs or Embedded Cores," in *Test Symposium, 2001. Proceedings. 10th Asian*, 2001, pp. 253–258.

[26] S. Guccione, P. Sundararajan, and S. McMillan, "Run-time Reconfigurable Testing of Programmable Logic Devices," Dec. 23 2003, uS Patent 6,668,237. [Online]. Available: https://www.google.com/patents/US6668237

[27] A. Jas and N. A. Touba, "Test Vector Decompression Via Cyclical Scan Chains and its Application to Testing Core-based Designs," in *Test Conference, 1998. Proceedings., International*, Oct 1998, pp. 458–464.

[28] R. Vaidyanathan and J. Trahan, *Dynamic Reconfiguration: Architectures and Algorithms*, 1st ed. Springer, Jan. 2004.

[29] H. M. El-Boghdadi, R. Vaidyanathan, J. L. Trahan, and S. Rai, "On the Communication Capability of the Self-Reconfigurable Gate Array Architecture," in *16th International Parallel and Distributed Processing Symposium (IPDPS*, 2002.

[30] H. M. El-Boghdadi, "Power-Aware Routing for Well-Nested Communications On The Circuit Switched Tree," in *21th International Parallel and Distributed Processing Symposium (IPDPS*, 2007, pp. 1–8.

[31] R. P. S. Sidhu, S. Wadhwa, A. Mei, and V. K. Prasanna, "A Self-Reconfigurable Gate Array Architecture," in *Field-Programmable Logic and Applications, The Roadmap to Reconfigurable Computing, 10th International Workshop, FPL 2000, Villach, Austria, August 27-30, 2000, Proceedings*, 2000, pp. 106–120.

[32] J. Mao, H. Zhou, H. Ye, and J. Lai, "FPGA bitstream compression and decompression using LZ and golomb coding," in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*. ACM, 2013, pp. 265–265.

[33] Z. Li and S. Hauck, "Configuration Compression for Virtex FPGAs," in *Field-Programmable Custom Computing Machines, 2001. FCCM '01. The 9th Annual IEEE Symposium on*, March 2001, pp. 147–159.

[34] C. Huriaux, A. Courtay, and O. Sentieys, "Design Flow and Run-Time Management for Compressed FPGA Configurations," in *DATE - Design, Automation and Test in Europe*, Mar. 2015. [Online]. Available: https://hal.inria.fr/hal-01089319

[35] B. Jacob, S. Ng, and D. Wang, *Memory Systems: Cache, DRAM, Disk*. Elsevier Science, 2010. [Online]. Available: https://books.google.com/books?id=SrP3aWedesC

[36] K. Roy, R. Vaidyanathan, and J. L. Trahan, "Routing Multiple Width Communications on the Circuit Switched Tree," *Int. J. Found. Comput. Sci.*, vol. 17, no. 2, pp. 271–286, 2006.

[37] H. P. Dharmasena and R. Vaidyanathan, "The Mesh With Binary Tree Networks: An Enhanced Mesh With Low Bus-Loading," *Journal of Interconnection Networks*, vol. 5, no. 2, pp. 131–150, 2004.

[38] Cadence Design Systems, Inc, "Encounter RTL Compiler, SoC Encounter RTL-to-GDSII System." [Online]. Available: https://www.cadence.com/en/default.aspx

[39] Oklahoma State University (OSU), North Carolina State University (NCSU), "FreePDK45: A Free OpenAccess 45nm PDK and Cell Library for Universities." [Online]. Available: http://vlsiarch.ecen.okstate.edu/flows/

[40] MATLAB, *version 7.10.0 (R2010a)*. Natick, Massachusetts: The MathWorks Inc., 2010.

[41] R. Kongari, "Cost and Performance Modeling of the MU-Decoder," Master's thesis, Louisiana State University, Baton Rouge, LA 70803, 2011.

Vita

Arash Ashrafi was born in March of 1990, in Tehran, Iran. He received his primary and secondary education from Rouzbeh High School in Tehran. After high school, Ashrafi received admission to the School of Electrical and Computer Engineering at the University of Tehran, the highest-ranked university in Iran. In December 2013, he received his Bachelor's degree in Electrical Engineering from University of Tehran. After his graduation, he came to the United States of America to pursue a master's degree in Electrical Engineering. In January 2014, he joined the Department of Electrical and Computer Engineering at Louisiana State University, Baton Rouge, Louisiana. He worked as graduate research assistant from September 2014 to May 2016 with the supervision of Dr. R. Vaidyanathan. He will attain his Master of Science degree in Electrical Engineering in the spring of 2016.